Contents

File Formats **(F)**

Intro

Introduction to file formats

Description

This section outlines the formats of various files. The C **struct** declarations for the file formats are given where applicable. Usually, these structures can be found in the directories **/usr/include** or **/usr/include/sys**. Note that include files are part of the Development System.

86rel

Intel 8086 Relocatable Format for Object Modules

Syntax

#include <sys/relsym86.h>

Description

Intel 8086 Relocatable Format, or *86rel*, is the object module format generated by *masm*(CP), and the input format for the linker *ld*(CP). The include file **relsym86.h** specifies appropriate definitions to access *86rel* format files from C. For the technical details of the *86rel* format, see *Intel 8086 Object Module Format External Product Specification*.

An *86rel* consists of one or more variable length records. Each record has at least three fields: the record type, length, and checksum. The first byte always denotes the record type. There are thirty-one different record types. Only eleven are used by *ld*(CP) and *masm*(CP). The word after the first byte is the length of the record in bytes, exclusive of the first three bytes. Following the length word are typically one or more fields. Each record type has a specific sequence of fields, some of which may be optional or of varying length. The very last byte in each record is a checksum. The checksum byte contains the sum modulo 256 of all other bytes in the record. The sum modulo 256 of all bytes in a record, including the checksum byte, should equal zero.

With few exceptions, *86rel* strings are length prefixed and have no trailing null. The first byte contains a number between 0 and 40, which is the remaining length of the string in bytes. Although the Intel specification limits the character set to upper case letters, digits, and the characters "?", "@", ":", ".", and "_", *masm*(CP) uses the complete ASCII character set.

The Intel Object Module Format (OMF) specification uses the term "index" to mean a positive integer either in the range 0 to 127, or 128 to 32,768. This terminology is retained in this document and elsewhere in the *86rel* literature. An index has one or two bytes. If the first byte has a leading 0 bit, the index is assumed to have only one byte, and the remainder of the byte represents a positive integer between 0 and 127. If the second byte has a leading 1 bit, the index is assumed to take up two bytes, and the remainder of the word represents a positive integer between 128 and 32,768.

Following is a list of record types and the hexadecimal value of their first byte, as defined in **relsym86.h**.

```
#define MRHEADR  0x6e /*rel module header/*
#define MREGINT      0x70 /*register initialization*/
#define MREDATA  0x72 /*explicit (enumerated) data image*/
#define MRIDATA      0x74 /*repeated (iterated) data image*/
#define MOVLDEF      0x76 /*overlay definition*/
#define MENDREC      0x78 /*block or overlay end record*/
#define MBLKDEF      0x7a /*block definition*/
#define MBLKEND      0x7c /*block end*/
#define MDEBSYM      0x7e /*debug symbols*/
#define MTHEADR  0x80 /*module header,
                         *usually first in a rel file*/
#define MLHEADR  0x82 /*link module header*/
#define MPEDATA  0x84 /*absolute data image*/
#define MPIDATA      0x86 /*absolute repeated (iterated)
                         *data image*/
#define MCOMENT  0x88 /*comment record*/
#define MMODEND  0x8a /*module end record*/
#define MEXTDEF      0x8c /*external definition*/
#define MTYPDEF      0x8e /*type definition*/
#define MPUBDEF      0x90 /*public definition*/
#define MLOCSYM      0x92 /*local symbols*/
#define MLINNUM      0x94 /*source line number*/
#define MLNAMES  0x96 /*name list record*/
#define MSEGDEF      0x98 /*segment definition*/
#define MGRPDEF      0x9a /*group definition*/
#define MFIXUPP      0x9c /*fix up previous data image*/
#define MNONE1   0x9e /*none*/
#define MLEDATA  0xa0 /*logical data image*/
#define MLIDATA      0xa2 /*logical repeated (iterated)
                         *data image*/
#define MLIBHED      0xa4 /*library header*/
#define MLIBNAM      0xa6 /*library names record*/
#define MLIBLOC      0xa8 /*library module locations*/
#define MLIBDIC      0xaa /*library dictionary*/
#define M386END      0x86 /*32 bit module end record*/
#define MPUB386      0x91 /*32 bit public definition*/
#define MLOC386      0x93 /*32 bit logical symbols*/
#define MLIN386      0x95 /*32 bit source line number*/
#define MSEG386      0x99 /*32 bit segment definition*/
#define MFIX386      0x9d /*fix up previous 32 bit data image*/
#define MLED386      0xa1 /*32 bit logical data image*/
#define MLID386      0xa3 /*32 bit logical repeated (iterated) data image*/
```

In the following discussion, the salient features of each record type are given. If the record is not used by either *masm*(CP) or *ld*(CP), it is not listed.

THEADR

The record type byte is 0x80. The THEADR record specifies the name of the source module at assembly-time (see Notes). The sole field is the T-MODULE NAME , which contains a length-prefixed string derived from the base name of the source module.

COMENT

The record type byte is 0x88. The COMENT record may contain a remark generated by the compiler system. *masm*(CP) inserts the string “UNIX 8086 ASSEMBLER .”

MODEND

The record type byte is 0x8a. The MODEND record terminates a module. It can specify whether the current module is to be used as the entry point to the linked executable. If the module is an entry point, the MODEND record can then specify the address of the entry point within the executable.

EXTDEF

The record type byte is 0x8c. The EXTDEF record contains the names and types of symbols defined in other modules by a PUBDEF record (see below). This corresponds to the C storage class “extern.” The fields consist of one or more length-prefixed strings, each with a following type index. The indices reference a TYPDEF record seen earlier in the module. *masm*(CP) generates only one EXTDEF per exterior symbol.

TYPDEF

The record type byte is 0x8e. The TYPDEF record gives a description of the type (size and storage attributes) of an object or objects. This description can then be referenced by EXTDEF , PUBDEF , and other records.

PUBDEF

The record type byte is 0x90. The PUBDEF record gives a list of one or more names that may be referenced by other modules at link-time (“publics”). The list of names is preceded by a group and segment index, which reference the location of the start of the list of publics within the current segment and group. If the segment and group indices are zero, a frame number is given to provide an absolute address in the module. The list consists of one or more of length-prefixed strings, each associated with a 16-bit offset within the current segment and a type index referring to a TYPDEF .

LNAMES

The record type byte is 0x96. The LNAMES record gives a series of length-prefixed strings which are associated with name indices within the current module. Each name is indexed in sequence given starting with 1. The names may then be referenced within the current module by successive SEGDEF and GRPDEF records to provide strings for segments, classes, overlays or groups.

SEGDEF

The record type byte is 0x98. The SEGDEF record provides an index to reference a segment, and information concerning segment addressing and attributes. This index may be used by other records to refer to the segment. The first byte in the record after the length field gives information about the alignment, and about combination attributes of the segment. The next word is the segment length in bytes. Note that this restrains segments to a maximum 65,536 bytes in length. Following this word is an index (see above) for the segment. Lastly, the SEGDEF may optionally contain class and/or overlay index fields.

GRPDEF

The record type is 0x9a. The GRPDEF record provides a name to reference several segments. The group name is implemented as an index (see above).

FIXUPP

The record byte is 0x9c. The FIXUPP record specifies one or more load-time address modifications (''fixups''). Each fixup refers to a location in a preceding LEDATA (see below) record. The fixup is specified by four data; a location, a mode, a target and a frame. The frame and target may be specified explicitly or by reference to an already defined fixup.

LEDATA

The record type byte is 0xa0. This record provides a contiguous text or data image which the loader *ld*(CP) uses to construct a portion of an 8086 run-time executable. The image might require additional processing (see FIXUPP) before being loaded into the executable. The image is preceded by two fields, a segment index and an enumerated data offset. The segment index (see INDEX) specifies a segment given by a previously seen SEGDEF . The enumerated data offset (a word) specifies the offset from the start of this segment.

See Also

as(CP), ld(CP)

Notes

If you attempt to load a number of modules assembled under the same basename, the loader will try to put them all in one big segment. In 286 programs, segment size is limited to 64K. In a large program the resulting segment size can easily exceed 64K. A large model code executable results from the link of one or more modules, composed of segments that aggregate into greater than 64K of text.

Hence, be sure that the assembly-time name of the module has the same basename as the source. This can occur if the source module is preprocessed not by *cc*(CP), but, for example, by hand or shell script, prior to assembly. The following example is incorrect:

```
#incorrect
cc -E module1.c | filter > x.c
cc x.c
mv x.o module1.o
cc -E module2.c | filter > x.c
cc x.c
mv x.o module2.o
cc -E module3.c | filter > x.c
cc x.c
mv x.o module3.o
ld module1.o module2.o module3.o
```

To avoid this, each of the modules should have a unique name when assembled, as follows:

```
#correct
cc -E module1.c | filter > x.c
cc -S x.c
mv x.s module1.s
as module1.s
.
.
.
ld module1.o module2.o module3.o
```

Value Added

86rel is an extension of AT&T System V provided by the Santa Cruz Operation.

a.out

UNIX common assembler and link editor output

Syntax

#include <a.out.h>

Description

The file name **a.out** is the default output file name from the link editor *ld*(CP). The link editor will make *a.out* executable if there were no errors in linking. The output file of the UNIX assembler *as (CP)* also follows the common object file format of the *a.out* file although the default file name is different.

A common object file consists of a file header, a UNIX system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
UNIX system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **-s** option of *ld*(CP) or if they were removed by *strip*(CP). Also note that the relocation information will be absent after linking unless the **-r** option of *ld*(CP) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. On your computer, the text segment starts at location virtual address 0.

The a.out file produced by *ld*(CP) may have one of two magic numbers in the first field of the UNIX system header. A magic number of 0410 indicates that the executable must be swapped through the private swapping store of the UNIX system, while the magic number 0413 causes the system to attempt to page the text directly from the a.out file.

In a 0410 executable, the text section is loaded at virtual location 0x00000000. The data section is loaded immediately following the end of the text section.

For a 0413 executable, the headers (file header, UNIX system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address will equal the sum of the sizes of the headers, and will vary depending on the number of sections in the a.out file. In an a.out file with 3 sections (.text, .data, and .bss) the first text address is at 0x000000D0. The data section starts in the next page table directory after the last one used by the text section, in the first page of that directory, with an offset into that page equal to the 1st unused memory offset in the last page of text. That is to say, given that *etext* is the address of the last byte of the text section, the 1st byte of the data section will be at 0x00400000 + (*etext* & 0xFFC00000) + ((*etext*+1) & 0xFFC00FFF).

On the 80386 computer the stack begins at location 7FFFFFFC and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the *brk*(S) system call.

For relocatable files the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an ''external symbol'', and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

File Header

The format of the **filehdr** header is:

```
struct filehdr
{
        unsigned short  f_magic;    /* magic number */
        unsigned short  f_nscns;    /* number of sections */
        long            f_timdat;   /* time and date stamp */
        long            f_symptr;   /* file ptr to symtab */
        long            f_nsyms;    /* # symtab entries */
        unsigned short  f_opthdr;   /* sizeof(opt hdr) */
        unsigned short  f_flags;    /* flags */
};
```

UNIX System Header

The format of the UNIX system header is:

```
typedef struct aouthdr
{
        short   magic;       /* magic number */
        short   vstamp;      /* version stamp */
        long    tsize;       /* text size in bytes, padded */
        long    dsize;       /* initialized data (.data) */
        long    bsize;       /* uninitialized data (.bss) */
        long    entry;       /* entry point */
        long    text_start;  /* base of text used for this file */
        long    data_start;  /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is:

```
struct scnhdr
{
        char            s_name[SYMNMLEN];/* section name */
        long            s_paddr;    /* physical address */
        long            s_vaddr;    /* virtual address */
        long            s_size;     /* section size */
        long            s_scnptr;   /* file ptr to raw data */
        long            s_relptr;   /* file ptr to relocation */
        long            s_lnnoptr;  /* file ptr to line numbers */
        unsigned short  s_nreloc;   /* # reloc entries */
        unsigned short  s_nlnno;    /* # line number entries */
        long            s_flags;    /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```
struct reloc
{
        long      r_vaddr;   /* (virtual) address of reference */
        long      r_symndx;  /* index into symbol table */
        ushort    r_type;    /* relocation type */
};
```

The start of the relocation information is *s_relptr* from the section header. If there is no relocation information, *s_relptr* is 0.

Symbol Table

The format of each symbol in the symbol table is

```
#define  SYMNMLEN 8
#define  FILNMLEN 14
#define  DIMNUM   4

struct syment
{
    union                               /* all ways to get a symbol name */
    {
        char       _n_name[SYMNMLEN]; /* name of symbol */
        struct
        {
            long   _n_zeroes;         /* == 0L if in string table */
            long   _n_offset;         /* location in string table */
        } _n_n;
        char       *_n_nptr[2];       /* allows overlaying */
    } _n;
    long           n_value;           /* value of symbol */
    short          n_scnum;           /* section number */
    unsigned short n_type;            /* type and derived type */
    char           n_sclass;          /* storage class */
    char           n_numaux;          /* number of aux entries */
};

#define  n_name    _n._n_name
#define  n_zeroes  _n._n_n._n_zeroes
#define  n_offset  _n._n_n._n_offset
#define  n_nptr    _n._n_nptr[1]
```

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent {
        struct {
                long   x_tagndx;
                union {
                        struct {
                                  unsigned short  x_lnno;
                                  unsigned short  x_size;
                        } x_lnsz;
                        long      x_fsize;
                } x_misc;
                union {
                        struct {
                                  long   x_lnnoptr;
                                  long   x_endndx;
                        }  x_fcn;
                        struct {
                                  unsigned short  x_dimen[DIMNUM];
                        } x_ary;
                } x_fcnary;
                unsigned short  x_tvndx;
        } x_sym;

        struct {
                char   x_fname[FILNMLEN];
        } x_file;

        struct {
                long        x_scnlen;
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        } x_scn;

        struct {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        } x_tv;
};
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is *f_symptr* (from the file header) bytes from the beginning of the file. If the symbol table is stripped, *f_symptr* is 0. The string table (if one exists) begins at *f_symptr* + (*f_nsyms* * SYMESZ) bytes from the beginning of the file.

See Also

as(CP), cc(CP), ld(CP), brk(S). filehdr(F), ldfcn(F), linenum(F), reloc(F), scnhdr(F), syms(F).

acct

format of per-process accounting file

Description

Files produced as a result of calling *acct*(S) have records in the form defined by **<sys/acct.h>.**

In *ac_flag*, the AFORK flag is turned on by each *fork*(S) and turned off by an *exec*(S). The *ac_comm* field is inherited from the parent process and is reset by any *exec*. Each time the system charges the process with a clock tick, it also adds the current process size to *ac_mem* computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of *ac_mem*/*ac_stime* can be viewed as an approximation to the mean process size, as modified by text-sharing.

See Also

acctcom(ADM), acct(S)

Notes

The *ac_mem* value for a short-lived command gives little information about the actual size of the command, because *ac_mem* may be incremented while a different command (e.g., the shell) is being executed by the process.

Standards Conformance

acct is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

ar

archive file format

Description

The archive command *ar* is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor *ld*(C).

A file produced by *ar* has a magic number at the start, followed by the constituent files, each preceded by a file header. The magic number is 0177545 octal (or 0xff65 hexadecimal). The header of each file is declared in **/usr/include/ar.h**.

Each file begins on a word boundary; a null byte is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

See Also

ar(CP), ld(CP)

archive

default backup device information

Description

/etc/default/archive contains information on system default backup devices for use by *sysadmsh*(ADM). The device entries are in the following format:

```
name=value  [name=value]  ...
```

value may contain white spaces if quoted, and newlines may be escaped with a backslash.

The following names are defined for */etc/default/archive*:

bdev	Name of the block interface device.
cdev	Name of the character interface device.
size	Size of the volume in either blocks or feet.
density	Volume density, such as 1600. If this value is missing or null, then *size* is in blocks; otherwise the *size* is in feet.
format	Command used to format the archive device.
blocking	Blocking factor.
desc	A description of the device, such as ‘‘Cartridge Tape.’’

See Also

sysadmsh(ADM)

authcap

authentication database

Syntax

/etc/auth/*
/tcb/files/auth/*

Description

The database contains authentication and identity information for users, kernels, and TCB files as well as system-wide parameters. It is intended to be used by programs to interrogate user and system values, as well as by authentication programs to update that information.

Structure of the Hierarchies

The complete database resides in two hierarchies: **/tcb/files/auth** and **/etc/auth**. The first hierarchy deals with user-specific files, and has subdirectories of one letter each of which is the starting letter for user name. Within each of these directories are regular files, each containing an *authcap*(F) format file for a particular user. Thus, all user names beginning with **x** have their respective authentication and identity information in a file in directory **/tcb/files/auth/x**.

The directories within **/etc/auth** contain system-wide information. The global system settings reside in the **/etc/auth/system** directory. The subsystem authorizations associated with each protected subsystem (a protected subsystem is privileged but does not require global authority to perform actions) are located in the **/etc/auth/subsystems** directory.

The following database files are contained in the **system** directory:

default	Default Control
files	File Control
ttys	Terminal Control
authorize	Primary and Secondary Authorization Control File
devassign	Device Assignment

A subsystem file name is the group name associated with the protected subsystem. The owner of all files is **auth** and the group is the group of the subsystem. Only the owner and group of this file may view the contents. The file **dflt_users** lists the users granted default subsystem authorizations.

Format of a File

Each data file in the hierarchy, whether system-wide or user-specific, has the same format. Each user file consists of one virtual line, optionally split into multiple physical lines with the '/' character present at the very end of all lines but the last. For instance, the line

```
blf:u_name=blf:u_id#16:u_encrypt=a78/a1.eitfn6:u_type=sso:chkent:
```

may be split into:

```
      blf:u_name=blf:u_id#16:\
              :u_encrypt=a78/a1.eitfn6:\
              :u_type=sso:chkent:
```

Note that all capabilities must be immediately preceded and followed with the ':' separator; multiple line entries require additional ones - one more per line. Multiple entries are separated by a newline:

```
drb:u_name=drb:u_id#75:u_maxtries#9:u_type=general:chkent:
blf:u_name=blf:u_id#76:u_maxtries#5:u_type=general:chkent:
```

For subsystem files, the file is a set of lines, each containing a user name terminated by a colon, followed by a comma-separated list of primary and secondary authorizations defined for that subsystem.

Format of a Line

The format of a line (except for subsystem files) is briefly as follows:

name|alt name(s)|description:cap1:cap2:cap3:...:capn:chkent:

The entry can be referenced by the name or any of the alternate names. A description field may document the entry. The entry name(s) and description are separated by the '|' character. The end of the name/description part of the entry is terminated by the ':' character. Alternate names and the description fields are optional.

At the end of each entry is the *chkent* field. This is used as an integrity check on each entry. The *authcap*(S) routines will reject all entries that do not have *chkent* at the very end.

Each entry has 0 or more capabilities, each terminated with the ':' character. Each capability has a unique name. Numeric capabilities have the format:

id#*num*

where num is a decimal or (0 preceded) octal number. Boolean capabilities have the format:

id

or

id@

where the first form signals the presence of the capability and the second form signals the absence of the capability. String capabilities have the format:

id=*string*

where string is 0 or more characters. The ‘\’ and ‘:’ characters are escaped as ‘\\’ and ‘\:’ respectively. Although it is not recommended, the same id may be used for different numeric, boolean, and string capabilities.

See Also

getprpwent(S), getdvagent(S), getprtcent(S), getprfient(S)

Value Added

authcap is an extension of AT&T System V provided by the Santa Cruz Operation.

checklist

list of file systems processed by fsck(ADM)

Description

The **/etc/checklist** file contains a list of the file systems to be checked when *fsck*(ADM) is invoked without arguments. The list contains at most 15 **special file** names. Each **special file** name must be on a separate line and must correspond to a file system.

See Also

fsck(ADM)

clock

the system real-time (time of day) clock

Description

The **clock** file provides access to the battery-powered, real-time time of day clock. Reading this file returns the current time; writing to the file sets the current time. The time, 10 bytes long, has the following form:

MMddhhmmyy

where *MM* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *yy* is the last two digits of the year. For example, the time:

0826150389 is 15:03 on August 26, 1989.

Files

/dev/clock

See Also

setclock(ADM)

Notes

Not all computers have battery-powered real-time time of day clocks. Refer to your computer's hardware reference manual.

core

format of core image file

Description

The Operating System writes out a core image of a terminated process when any of various errors occur. See *signal*(S) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The core image is called *core* and is written in the process' working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The first section of the core image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter *usize*, which is defined in **/usr/include/sys/param.h**. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in **/usr/include/sys/user.h**. The locations of registers, are outlined in **/usr/include/sys/reg.h**.

See Also

adb(CP), setuid(S), signal(S)

cpio

format of cpio archive

Description

The *header* structure, when the **-c** option of *cpio*(C) is not used, is:

```
struct {
               short     h_magic,
                         h_dev;
               ushort    h_ino,
                         h_mode,
                         h_uid,
                         h_gid;
               short     h_nlink,
                         h_rdev,
                         h_mtime[2],
                         h_namesize,
                         h_filesize[2];
               char      h_name[h_namesize rounded to word];
} Hdr;
```

When the **-c** option is used, the *header* information is described by:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
          &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
          &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
          &Longtime, &Hdr.h_namesize,&Longfile,Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The contents of each file are recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in *stat*(S). The length of the null-terminated path name *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with *h_filesize* equal to zero.

See Also

cpio(C), find(C), stat(S)

Standards Conformance

cpio is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

default

default program information directory

Description

The files in the directory **/etc/default** contain the default information used by system commands such as **xbackup**(ADM) and **remote**(C). Default information is any information required by the command that is not explicitly given when the command is invoked.

The directory may contain zero or more files. Each file corresponds to one or more commands. A command searches a file whenever it has been invoked without sufficient information. Each file contains zero or more entries which define the default information. Each entry has the form:

```
keyword
```

or

```
keyword=value
```

where *keyword* identifies the type of information available and *value* defines its value. Both *keyword* and *value* must consist of letters, digits, and punctuation. The exact spelling of a *keyword* and the appropriate *values* depend on the command and are described with the individual commands.

Any line in a file beginning with a number sign (#) is considered a comment and is ignored.

Files

/etc/default/*

See Also

archive(F), xbackup(ADM), boot(HW), cron(C), dos(C), dumpdir(C), filesys(F), login(M), lpr(C), mapchan(M), mapchan(F), micnet (F), authsh(ADM) remote(C), xrestore(ADM), su(C), tar(C)

Note

Not all commands use **/etc/default** files. Please refer to the manual page for a specific command to determine if **/etc/default** files are used, and what information is specified.

Value Added

default is an extension of AT&T System V provided by the Santa Cruz Operation.

devices

format of UUCP devices file

Description

The *Devices* file (**/usr/lib/uucp/Devices**) contains information for all the devices that can be used to establish a link to a remote computer. These devices include automatic call units, direct links, and network connections. This file works closely with the **Dialers**, **Systems**, and **Dialcodes** files.

Each entry in the *Devices* file has the following format:

type ttyline dialerline speed dialer-token

where:

type can contain one of two keywords (**direct** or **ACU**), the name of a Local Area Network switch, or a system name.

ttyline contains the device name of the line (port) associated with the *Devices* entry. For example, if the Automatic Dial Modem for a particular entry is attached to the **/dev/tty11** line, the name entered in this field is **tty11**.

dialerline is useful only for 801 type dialers, which do not contain a modem and must use an additional line. If you do not have an 801 dialer, enter a hyphen (-) as a placeholder.

speed is the speed or speed range of the device. It may contain an indicator for distinguishing different dialer classes.

dialer-token contains pairs of dialers and tokens. Each represents a dialer and an argument to be passed to it. The *dialer* portion can be the name of an automatic dial modem, or it may be a **direct** for a direct link device.

For best results, dialer programs are preferred over **Dialers** entries. The following entry is an example of an entry using a dialer binary:

```
ACU  ttynn  -  300-2400  /usr/lib/uucp/dialHA24
```

Note all lines must have at least 5 fields. Use "-" for unused fields. Types that appear in the 5th field must be either built-in functions (801, Sytek, TCP, Unetserver, DK) or standard functions whose name appears in the first field in the **Dialers** file.

Two escape characters can be used in this file:

\D   which means don't translate the phone /token

\T   translate the phone /token using the Dialcodes file

Both refer to the phone number field in the **Systems** file (field 5). \D should always be used with entries in the **Dialers** file, since the **Dialers** file can contain a T to expand the number if necessary. \T should only be used with built-in functions that require expansion.

Note that if a phone number is expected and a \D or \T is not present a \T is used for a built-in, and \D is used for an entry referencing the **Dialers** file.

Examples

The following are examples of common **Devices** files.

Standard modem line

```
ACU tty00 - 1200 801
ACU tty00 - 1200 penril
or
ACU tty00 - 1200 penril \D
```

A direct line

This example will allow **cu -ltty00** to work. This entry could also be used for certain modems in manual mode.

```
Direct tty00 - 4800 direct
```

A ventel modem on a develcon switch

"vent" is the token given to the develcon to reach the ventel modem.

```
ACU tty00 - 1200 develcon vent ventel
ACU tty00 - 1200 develcon vent ventel \D
```

To reach a system on the local develcon switch

```
Develcon tty00 - Any develcon \D
```

A direct connection to a system

```
systemx tty00 - Any direct
```

STREAMS Network Examples

A STREAMS network that conforms to the AT&T Transport Interface with a direct connection to login service (i.e., without explicitly using the Network Listener Service dial script):

```
networkx,eg devicex - - TLIS \D
```

The **Systems** file entry looks like:

```
systemx Any networkx - addressx in:--in: nuucp word: nuucp
```

You must replace *systemx*, *networkx*, *addressx*, and *devicex* with system name, network name, network address and network device, respectively. For example, entries for machine ''sffo'' on a STARLAN NETWORK might look like:

```
sffoo Any STARLAN - sffoo in:--in: nuucp word: nuucp
```

and:

```
STARLAN,eg starlan - - TLIS \D
```

To use a STREAMS network that conforms to the AT&T Transport Interface and that uses the Network Listener Service dial script to negotiate for a server:

```
networkx,eg devicex - - TLIS \D nls
```

To use a non-STREAMS network that conforms to the AT&T Transport Interface and that uses the Network Listener Service dial script to negotiate for a server:

```
networkx,eg devicex - - TLI \D nls
```

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C), dialers(F)

Notes

Blank lines and lines that begin with a <space>, <tab>, or are ignored. protocols can be specified as a comma-subfield of the device type either in the **Devices** file (where device type is field 1) or in the **Systems** file (where it is field 3).

dialcodes

format of UUCP Dialcode abbreviations file

Description

The **Dialcodes** file (**/usr/lib/uucp/Dialcodes**) contains the Dialcode abbreviations that can be used in the *Phone* field of the **Systems** file. This feature allows you to create a standard **Systems** file for distribution among several sites that have different phone systems and area codes.

If two remote sites in a network need to link with the same sites, but have different internal phone systems each site can share the same **Systems** file, but have different entries in a **Dialcodes** file. Each entry has the following format:

abb dial-seq

where:

abb is the abbreviation used in the **Systems** file *phone* field

dial-seq is the dial sequence that is passed to the dialer when that particular **Systems** file entry is accessed.

The following entry would be set up to work with a *phone* field in the *Systems* file such as *jt7867 :*

```
jt 9=847-
```

When the entry containing *jt7867* is encountered, the following sequence is sent to the dialer if the token in the dialer-token-pair is **\T** :

```
9=847-7867
```

The phone number is made up of an optional alphabetic abbreviation and a numeric part. If an abbreviation is used, it must be one that is listed in the *Dialcodes* file.

```
NY 9=1212555
```

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C), systems(F)

dialers

format of UUCP Dialers file

Description

The **Dialers** file (**/usr/lib/uucp/Dialers**) specifies the initial conversation that must take place on a line before it can be made available for transferring data. This conversation is usually a sequence of ASCII strings that is transmitted and expected, and it is often used to dial a phone number using an ASCII dialer (such as the Automatic Dial Modem).

A modem that is used for dialing in and out may require a second **Dialers** entry. This is to reinitialize the line to dial-in after it has been used for dial-out. The name of the dial-in version of a dialer must begin with an ampersand. For example, the **Dialers** file contains a **hayes2400** and a **&hayes2400** entry.

The fifth field in a **Devices** file entry is an index into the **Dialers** file or a special dialer type. Here an attempt is made to match the fifth field in the **Devices** file with the first field of each **Dialers** file entry. In addition, each odd numbered **Devices** field starting with the seventh position is used as an index into the **Dialers** file. If the match succeeds, the **Dialers** entry is interpreted to perform the dialer negotiations. Each entry in the **Dialers** file has the following format:

dialer substitutions expect-send ...

The *dialer* field matches the fifth and additional odd numbered fields in the **Devices** file. The *substitutions* field is a translate string: the first of each pair of characters is mapped to the second character in the pair. This is usually used to translate = and - into whatever the dialer requires for ‘‘wait for dialtone’’ and ‘‘pause.’’

The remaining *expect-send* fields are character strings. Below are some character strings distributed with the UUCP package in the **Dialers** file.

Dialers file entries

```
penril     =W-P "" \d > s\p9\c )-W\p\r\ds\p9\c-) y\c : \E\TP > 9\c OK
ventel     =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
hayes      =,-, "" \dAT\r\c OK\r \EATDT\T\r\c CONNECT
rixon      =&-% "" \d\r\r\c $ s9\c )-W\r\ds9\c-) s\c : \T\r\c $ 9\c LINE
vadiac     =K-K "" \005\p *-\005\p-*\005\p-* D\p BER? \E\T\e \r\c LINE
develcon   "" "" \pr\ps\c est:\007 \E\D\e \007
micom      ""    "" \s\c NAME? \D\r\c GO
direct
att2212c   =+-,    "" \r\c :--: ato12=y,T\T\r\c red
att4000    =,-,    "" \033\r\r\c DEM: \033s0401\c \006 \033s0901\c \
               \006 \033s1001\c \006 \033s1102\c \006 \033dT\T\r\c \006
att2224    =+-,    "" \r\c :--: T\T\r\c red
nls            ""       "" NLPS:000:001:1\N\c
```

The meaning of some of the escape characters (those beginning with '‘\'') used in the **Dialers** file are listed below:

`\p`	pause (approximately ¼ to ½ second)
`\d`	delay (approximately 2 seconds)
`\D`	phone number or token without **Dialcodes** translation
`\T`	phone number or token with **Dialcodes** translation
`\K`	insert a BREAK
`\E`	enable echo checking (for slow devices)
`\e`	disable echo checking
`\r`	carriage return
`\c`	no new-line or carriage return
`\n`	send new-line
`\nnn`	send octal number.

Additional escape characters that may be used are listed in the section discussing the **Systems** file.

The penril entry in the **Dialers** file is executed as follows. First, the phone number argument is translated, replacing any = with a **W** (wait for dialtone) and replacing any - with a **P** (pause). The handshake given by the remainder of the line works as follows:

`""`	Wait for nothing.
`\d`	Delay for 2 seconds.
`>`	Wait for a **>**.

`s\p9\c` Send an **s**, pause for ½ second, send a **9**, send no terminating new-line

`)-W\p\r\ds\p9\c-)` Wait for a **)**. If it is not received, process the string between the **-** characters as follows. Send a **W**, pause, send a carriage-return, delay, send an **s**, pause, send a **9**, without a new-line, and then wait for the **)**.

`y\c` Send a **y**.

`:` Wait for a **:**.

`\E\TP` Enable echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.) Then, send the phone number. The **\T** means take the phone number passed as an argument and apply the **Dialcodes** translation and the modem function translation specified by field 2 of this entry. Then send a **P**.

`>` Wait for a **>**.

`9\c` Send a **9** without a new-line.

`OK` Waiting for the string **OK**.

See Also

dial(ADM), uucico(ADM), uucp(C), uux(C), uuxqt(C), devices(F)

Notes

Dialer binaries (located in **/usr/lib/uucp**) are preferred over **Dialers** entries. Binaries are more reliable. Refer to the *dial* man page for more information on creating your own dialer binaries.

dir

format of a directory

Syntax

#include <sys/dir.h>

Description

A directory behaves exactly like an ordinary file, except that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry (see *filesystem* (F)). The structure of a directory is given in the include file **/usr/include/sys/dir.h**.

By convention, the first two entries in each directory are‘‘dot’’ (**.**) and ‘‘dotdot’’ (**..**). The first is an entry for the directory itself. The second is for the parent directory. The meaning of dotdot is modified for the root directory of the master file system; there is no parent, so dotdot has the same meaning as dot.

See Also

filesystem(F)

dirent

filesystem-independent directory entry

Syntax

#include <sys/types.h>
#include <sys/dirent.h>

Description

Different file system types may have different directory entries. The *dirent* structure defines a file-system-independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the *getdents*(S) system call.

The *dirent* structure is defined below.

```
struct  dirent {
                    long                 d_ino;
                    off_t                d_off;
                    unsigned short       d_reclen;
                    char                 d_name[1];
            };
```

The *d_ino* is a number which is unique for each file in the file system. The field *d_off* is the offset of that directory entry in the actual file system directory. The field *d_name* is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most MAXNAMLEN characters. This results in file system independent directory entries being variable length entities. The value of *d_reclen* is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that it will always result in the next entry being on a long boundary.

Files

/usr/include/sys/dirent.h

See Also

getdents(S)

Standards Conformance

dirent is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

filehdr

file header for common object files

Syntax

#include <filehdr.h>

Description

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct  filehdr
{
        unsigned short  f_magic ;   /* magic number */
        unsigned short  f_nscns ;   /* number of sections */
        long            f_timdat ;  /* time & date stamp */
        long            f_symptr ;  /* file ptr to symtab */
        long            f_nsyms ;   /* # symtab entries */
        unsigned short  f_opthdr ;  /* sizeof(opt hdr) */
        unsigned short  f_flags ;   /* flags */
} ;
```

F_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek*(S) to position an I/O stream to the symbol table. The UNIX system optional header is 28-bytes. The valid magic numbers are given below:

```
#define I286SMAGIC 0512 /* 80286 computers—small model
                           programs */
#define I286LMAGIC 0522 /* 80286 computers—large model
                           programs */
#define I386MAGIC  0514 /* 80386 computers */
#define FBOMAGIC   0560 /* 3B2 and 3B15 computers */
#define N3BMAGIC   0550 /* 3B20 computer */
#define NTVMAGIC   0551 /* 3B20 computer */

#define VAXWRMAGIC 0570 /* VAX writable text segments */
#define VAXROMAGIC 0575 /* VAX read only sharable
                           text segments */
```

The value in *f_timdat* is obtained from the *time*(S) system call. Flag bits currently defined are:

```
#define F_RELFLG   0000001 /* relocation entries stripped */
#define F_EXEC     0000002 /* file is executable */
```

```
#define F_LNNO    0000004 /* line numbers stripped */
#define F_LSYMS   0000010 /* local symbols stripped */
#define F_MINMAL  0000020 /* minimal object file */
#define F_UPDATE  0000040 /* update file, ogen produced */
#define F_SWABD   0000100 /* file is "pre-swabbed" */
#define F_AR16WR  0000200 /* 16-bit DEC host */
#define F_AR32WR  0000400 /* 32-bit DEC host */
#define F_AR32W   0001000 /* non-DEC host */
#define F_PATCH   0002000 /* "patch" list in opt hdr */
#define F_80186   010000  /* contains 80186 instructions */
#define F_80286   020000  /* contains 80286 instructions */
#define F_BM32ID  0160000 /* WE32000 family ID field */
#define F_BM32B   0020000 /* file contains WE 32100 code */
#define F_BM32MAU 0040000 /* file reqs MAU to execute */
#define F_BM32RST 0010000 /* this object file contains restore
                             work around [3B15/3B2 only] */
```

See Also

time(S), fseek(S), a.out(F)

filesys

default information for mounting filesystems

Description

/etc/default/filesys contains information for mounting filesystems in the following format:

```
name=value  [name=value] ...
```

value may contain white spaces if quoted, and newlines may be escaped with a backslash.

mnt (see *mnt*(C)) and *sysadmsh*(ADM) use the information in the */etc/default/filesys* when the system comes up multiuser. The following names are defined for */etc/default/filesys*:

bdev
Name of the block interface device.

cdev
Name of the character interface device.

size
Size in blocks.

mountdir
Directory on which the filesystem is mounted.

desc
A description of the filesystem. For example, “User filesystem."

mountflags
Any flags passed to the **mount**(ADM) command.

fsckflags
Any flags passed to the **fsck**(ADM) command.

rcmount
Whether or not to mount the filesystem when the system goes multiuser. Can be “yes”, “no” or “prompt”. If set to “prompt”, you are prompted when it is time to mount the filesystem.

See Also

mount(ADM), mnt(C), sysadmsh(ADM)

filesystem

format of filesystem types

Syntax

#include <sys/fs/??filsys.h>
#include <sys/types.h>
#include <sys/param.h>

Description

Every filesystem storage volume (for example, a hard disk) has a common format for certain vital information. Every such volume is divided into a certain number of 1024-byte blocks. There are four fileystem types available:

S51K (UNIX fileystem)
XENIX
AFS (ACER Fast Filesystem)
DOS

The DOS filesystem is a 512-byte filesystem. (The DOS filesystem structure is shown in **/usr/include/fs/ dosfilsys.h**. This page does not discuss the format of the DOS filesystem in detail. Consult a DOS reference for more information.)

Block 0 is unused and is available to contain a bootstrap program or other information.

Block 1 is the *super-block*. The format of the S51K, AFS, and XENIX filesystem super-blocks are described in two files in the directory **/usr/include/fs**: **s5filsys.h** (S51K and AFS), **xxfilsys.h** (XENIX). The XENIX filesystem boot block is 1024-bytes; the S51K and UNIX boot blocks are 512-byte blocks. In these include files, *s_isize* is the address of the first data block after the i-list. The i-list starts just after the super-block in block 2; thus the i-list is *s_isize*-2 blocks long. *s_fsize* is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers. If an ''impossible'' block number is allocated from the free list or is freed, a diagnostic is written on the console. Moreover, the free array is cleared so as to prevent further allocation from a presumably corrupted free list.

The free list for S51K and XENIX volumes (but not AFS) is maintained as follows: The *s_free* array contains, in *s_free*[1], ..., *s_free*[*s_nfree*-1], up to NICFREE-1 numbers of free blocks. *s_free*[0] is the block number of the head of a chain of blocks constituting the free list. The first short in each free-chain block is the number (up to NICFREE) of free-block numbers listed in the next NICFREE longs of this chain member. The first of these NICFREE blocks is the link to the next member of the chain. To allocate a block: decrement *s_nfree*, and the new block is *s_free*[*s_nfree*]. If the new block number is 0, there are no blocks left, so give an error. If *s_nfree* becomes 0, read in the block named by the new block number, replace *s_nfree* by its first word, and copy the block numbers in the next NICFREE longs into the *s_free* array. To free a block, check if *s_nfree* is NICFREE; if so, copy *s_nfree* and the *s_free* array into it, write it out, and set *s_nfree* to 0. In any event set *s_free*[*s_nfree*] to the freed block's number and increment *s_nfree*.

In the AFS filesystem, the free list is maintained differently, The AFS freelist is organized as a bitmap, one bit per (1K) block in the filesystem. This organization makes it easy to find contiguous stretches of free blocks.

s_tfree is the total free blocks available in the filesystem.

s_ninode is the number of free i-numbers in the *s_inode* array. To allocate an inode: if *s_ninode* is greater than 0, decrement it and return *s_inode*[*s_ninode*]. If it was 0, read the i-list and place the numbers of all free inodes (up to NICINOD) into the *s_inode* array, then try again. To free an inode, provided *s_ninode* is less than NICINOD, place its number into *s_inode*[*s_ninode*] and increment *s_ninode*. If *s_ninode* is already NICINOD, do not bother to enter the freed inode into any table. This list of inodes only speeds up the allocation process. The information about whether the inode is really free is maintained in the inode itself.

s_tinode is the total free inodes available in the file system.

The following applies only to S51K and AFS fileystems: *s_state* indicates the state of the file system. A cleanly unmounted, not damaged file system is indicated by the FsOKAY state. After a file system has been mounted for update, the state changes to FsACTIVE. A special case is used for the root file system. If the root file system appears damaged at boot time, it is mounted but marked FsBAD. Lastly, after a file system has been unmounted, the state reverts to FsOKAY.

s_flock and *s_ilock* are flags maintained in the core copy of the filesystem while it is mounted and their values on disk are immaterial. The value of *s_fmod* on disk is also immaterial, and is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

s_ronly is a read-only flag to indicate write-protection.

s_time is the last time the super-block of the file system was changed, and is a double precision representation of the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). During a reboot, the *s_time* of the super-block for the root file system is used to set the system's idea of the time.

I-numbers begin at 1, and the storage for inodes begins in block 2. Also, inodes are 64 bytes long, so 16 of them fit into a block. Therefore, inode i is located in block $(i+31)/16$, and begins $64\times((i+31) \pmod{16})$ bytes from its start. Inode 1 is reserved for future use. Inode 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each inode represents one file. For the format of an inode and its flags, see *inode*(F).

Files

/usr/include/sys/filsys.h

/usr/include/sys/stat.h

See Also

fsck(ADM), mkfs(ADM), inode(F)

fspec

format specification in text files

Description

It is sometimes convenient to maintain text files on the UNIX system with non-standard tabs, (i.e., tabs which are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by UNIX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of *tabs* must be one of the following:

1. a list of column numbers separated by commas, indicating tabs set at the specified columns;
2. a **-** followed immediately by an integer *n*, indicating tabs at intervals of *n* columns;
3. a **-** followed by the name of a ''canned'' tab specification.

Standard tabs are specified by **t-8**, or equivalently, **t1,9,17,25,** etc. The canned tabs which are recognized are defined by the *tabs*(C) command.

s*size* The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t-8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

See Also

ed(C), newform(C), tabs(C)

gettydefs

speed and terminal settings used by getty

Description

The **/etc/gettydefs** file contains information used by *getty*(M) to set up the speed and terminal settings for a line. It supplies information on what the *login* prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a BREAK character.

Each entry in **/etc/gettydefs** has the following format:

label# initial-flags # final-flags # login-prompt #next-label [# login-program]

Each entry must be followed by a carriage return and a blank line. The various fields can contain quoted characters of the form **\b**, **\n**, **\c**, etc., as well as \nnn, where *nnn* is the octal value of the desired character. The various fields are:

label This is the string against which *getty*(M) tries to match its second argument. It is often the speed, such as **1200**, at which the terminal is supposed to run, but it need not be (see below).

initial-flags These flags are the initial *ioctl*(S) settings to which the terminal is to be set if a terminal type is not specified to *getty*(M). The flags that *getty*(M) understands are the same as the ones listed in **/usr/include/sys/termio.h** [see *termio*(M)]. Normally only the speed flag is required in the *initial-flags*. *getty*(M) automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until *getty*(M) executes *login*(M).

final-flags Sets the same values as the *initial-flags*. These flags are set just prior to *getty* executing *login-program*. The speed flag is again required. The composite flag **SANE** is a composite flag that sets the following *termio*(M) parameters:

modes set:
CREAD BRKINT IGNPAR ISTRIP ICRNL IXON ISIG ICANON ECHO ECHOK OPOST ONLCR

modes cleared:
CLOCAL IGNBRK PARMRK INPCK INLCR IUCLC IXOFF XCASE ECHOE ECHONL NOFLSH OLCUC OCRNL ONOCR ONLRET OFILL OFDEL NLDLY CRDLY TABDLY BSDLY VTDLY FFDLY

The other two commonly specified *final-flags* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.

login-prompt Contains login prompt message that greets users. Unlike the above fields where white space is ignored (a space, tab, or new-line), it is included in the *login-prompt* field. The '@' in the login-prompt field is expanded to the first line (or second line if it exists) in **/etc/systemid** (unless the '@' is preceded by a '\'). Several character sequences are recognized, including:

\n	Linefeed
\r	Carriage return
\v	Vertical tab
\nnn	(3 octal digits) Specify ASCII character
\t	Tab
\f	Form feed
\b	Backspace

next-label Identifies the next entry in *gettydefs* for *getty* to try if the current one is not successful. *Getty* tries the next label if a user presses the BREAK key while attempting to log in to the system. Groups of entries, for example, for dial-up lines or for TTY lines, should form a closed set so that *getty* cycles back to the original entry if none of the entries is successful. For instance, **2400** linked to **1200**, which in turn is linked to **300**, which finally is linked to **2400**.

login-program
The name of the program that actually logs the user onto UNIX. The default program is **/etc/login**. If preceded by the keyword **AUTO**, *getty* will not prompt for a username, but instead uses its first argument as the username and executes the *login-program* immediately.

If *getty* is called without a second argument, then the first entry of **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. The first entry is also used if *getty* can not find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into the command which will bring up a terminal at **300** baud.

After modifying **/etc/gettydefs**, run it through *getty* with the check option to be sure there are no errors.

Files

/etc/gettydefs

See Also

stty(C), ioctl(S), getty(M), login(M)

gps

graphical primitive string, format of graphical files

Description

GPS is a format used to store graphical data. Several routines have been developed to edit and display GPS files on various devices. Also, higher level graphics programs such as *plot* [in *stat*(1G)] and *vtoc* [in *toc*(1G)] produce GPS format output files.

A GPS is composed of five types of graphical data or primitives.

GPS PRIMITIVES

lines The *lines* primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a *move* to that location. (A *move* is a relocation of the graphic cursor without drawing.) Successive points produce line segments from the previous point. Parameters are available to set *color*, *weight*, and *style* (see below).

arc The *arc* primitive has a variable number of points to which a curve is fit. The first point produces a *move* to that point. If only two points are included, a line connecting the points will result; if three points a circular arc through the points is drawn; and if more than three, lines connect the points. (In the future, a spline will be fit to the points if they number greater than three.) Parameters are available to set *color, weight,* and *style*.

text The *text* primitive draws characters. It requires a single point which locates the center of the first character to be drawn. Parameters are *color*, *font*, *textsize*, and *textangle*.

hardware The *hardware* primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the *hardware* string.

comment A *comment* is an integer string that is included in a GPS file but causes nothing to be displayed. All GPS files begin with a comment of zero length.

GPS PARAMETERS

color *Color* is an integer value set for *arc, lines,* and *text* primitives.

weight *Weight* is an integer value set for *arc* and *lines* primitives to indicate line thickness. The value **0** is narrow weight, **1** is bold, and **2** is medium weight.

style *Style* is an integer value set for *lines* and *arc* primitives to give one of the five different line styles that can be drawn on TEKTRONIX 4010 series storage tubes. They are:

0	solid
1	dotted
2	dot dashed
3	dashed
4	long dashed

font An integer value set for *text* primitives to designate the text font to be used in drawing a character string. (Currently *font* is expressed as a four-bit *weight* value followed by a four-bit *style* value.)

textsize *Textsize* is an integer value used in *text* primitives to express the size of the characters to be drawn. *Textsize* represents the height of characters in absolute *universe-units* and is stored at one-fifth this value in the size-orientation (*so*) word (see below).

textangle *Textangle* is a signed integer value used in *text* primitives to express rotation of the character string around the beginning point. *Textangle* is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (*so*) word as a value 256/360 of it's absolute value.

ORGANIZATION

GPS primitives are organized internally as follows:

lines	*cw points sw*
arc	*cw points sw*
text	*cw point sw so* [*string*]
hardware	*cw point* [*string*]
comment	*cw* [*string*]

cw *cw* is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word-count for that primitive.

point(s) *Point(s)* is one or more pairs of integer coordinates. *Text* and *hardware* primitives only require a single *point*. *Point(s)* are values within a Cartesian plane or *universe* having 64K (-32K to +32K) points on each axis.

sw *Sw* is the style-word and is used in *lines, arc,* and *text* primitives. For all three, eight bits contain *color* information. In *arc* and *lines* eight bits are divided as four bits *weight* and four bits *style*. In the *text* primitive eight bits of *sw* contain the *font*.

so *So* is the size-orientation word used in *text* primitives. Eight bits contain text size and eight bits contain text rotation.

string *String* is a null-terminated character string. If the string does not end on a word boundary, an additional null is added to the GPS file to insure word-boundary alignment.

See Also

graphics(ADM), stat(ADM), toc(ADM)

group

format of the group file

Description

group contains the following information for each group:

- Group name

- Numerical group ID

- Comma-separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a newline. If the password field is null, no password is demanded.

This file resides in directory **/etc**. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group IDs to names.

Files

/etc/group

See Also

newgrp(C), passwd(C), passwd(F)

Standards Conformance

group is conformant with:

The X/Open Portability Guide II of January 1987;
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

hs

High Sierra/ISO-9660 CD-ROM filesystem

Description

The hs filesystem module supports the mounting of CD-ROM filesystems conforming to the High Sierra/ISO-9660 format.

Files

/usr/include/sys/fs/hs*

See Also

cdrom(HW), mkdev(ADM), mount(ADM)

Notes

Since the CD-ROM is a read-only device it is only possible to mount CD-ROM filesystems as read-only. The kernel enforces this regardless of whether the **-r** option of *mount*(ADM) was used when the filesystem was mounted.

The command *mkdev high-sierra* can be used to interactively configure High Sierra/ISO-9660 filesystem support.

inittab, init.base

script for the init process

Description

The **inittab** file supplies the script to *init*'s role as a general process dispatcher. The process that constitutes the majority of *init*'s process dispatching activities is the line process **/etc/getty** that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

The **inittab** file is recreated automatically by *idmkinit* at boot time anytime the kernel has been reconfigured. To construct a new **inittab** file, *idmkinit* reads the file **/etc/conf/cf.d/init.base**, which contains a base set of **inittab** entries that are required for the system, and combines these base entries with add-on entries from the device driver init files in the directory **/etc/conf/init.d**.

If you add an entry directly to **inittab**, the change exists only until the kernel is relinked. To add an entry permanently, you must also edit **/etc/conf/cf.d/init.base**. The **init.base** file has the same format as **inittab**.

The *inittab* file is composed of entries that are position-dependent and have the following format:

```
id:rstate:action:process
```

Each entry is delimited by a new-line; however, a backslash (\) preceding a new-line indicates a continuation of the entry. Up to 512 characters per entry are permitted. Comments may be inserted in the *process* field using the *sh*(C) convention for comments. Comments for lines that spawn *getty*s are displayed by the *who*(C) command. It is expected that they will contain some information about the line such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the **inittab** file. The entry fields are:

id This is up to four characters used to uniquely identify an entry.

rstate This defines the *run-level* in which this entry is to be processed. *Run-levels* effectively correspond to a configuration of processes in the system. That is, each process spawned by *init* is assigned a *run-level* or *run-levels* in which it is allowed to exist. The *run-levels* are represented by a number ranging from **0** through **6**. As an example, if the system is in *run-level* **1**, only those entries having a **1** in the *rstate* field *levels*, all processes which do not have an entry in the *rstate* field for the target *run-level* will be sent the warning signal (**SIGTERM**) and allowed a 20-second grace period before being forcibly terminated by a kill signal (**SIGKILL**). The *rstate* field can define multiple *run-levels* for a process by selecting more than one *run-level* in any combination from **0-6**. If no *run-level* is specified, then the process is assumed to be valid at all *run-levels* **0-6**. There are three other values, **a**, **b**, and **c**, which can appear in the *rstate* field, even though they are not true *run-levels*. Entries which have these characters in the *rstate* field are processed only when the *telinit* [see *init*(M)] process requests them to be run (regardless of the current *run-level* of the system). They differ from *run-levels* in that *init* can never enter *run-level* **a**, **b**, or **c**. Also, a request for the execution of any of these processes does not change the current *run-level*. Furthermore, a process started by an **a**, **b**, or **c** command is not killed when *init* changes levels. They are only killed if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab**, or *init* goes into the *SINGLE USER* state.

action Key words in this field tell *init* how to treat the process specified in the *process* field. The actions recognized by *init* are as follows:

respawn If the process does not exist, then start the process, do not wait for its termination (continue scanning the **inittab** file); and when it dies, restart the process. If the process currently exists, then do nothing and continue scanning the **inittab** file.

wait Upon *init*'s entering the *run-level* that matches the entry's *rstate*, start the process and wait for its termination. All subsequent reads of the **inittab** file while *init* is in the same *run-level* will cause *init* to ignore this entry.

once Upon *init*'s entering a *run-level* that matches the entry's *rstate*, start the process, do not wait for its termination. When it dies, do not restart the process. If upon entering a new *run-level*, where the process is still running from a previous *run-level* change, the program will not be restarted.

boot The entry is to be processed only at *init*'s boot-time read of the **inittab** file. *init* is to start the process, not wait for its termination; and when it dies, not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match *init*'s *run-level* at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait The entry is to be processed the first time *init* goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process will run right after the boot.) *init* starts the process, waits for its termination and, when it dies, does not restart the process.

powerfail Execute the process associated with this entry only when *init* receives a power fail signal [**SIGPWR** see *signal*(S)].

powerwait Execute the process associated with this entry only when *init* receives a power fail signal (**SIGPWR**) and wait until it terminates before continuing any processing of **inittab**.

off If the process associated with this entry is currently running, send the warning signal (**SIGTERM**) and wait 20 seconds before forcibly terminating the process via the kill signal (**SIGKILL**). If the process is nonexistent, ignore the entry.

ondemand This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with *run-levels*. This is used only with the **a**, **b**, or **c** values described in the *rstate* field.

initdefault An entry with this *action* is only scanned when *init* initially invoked. *init* uses this entry, if it exists, to determine which *run-level* to enter initially. It does this by taking the highest *run-level* specified in the **rstate** field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **0123456** and so *init* will enter *run-level* **6**. Additionally, if *init* does not find an **initdefault** entry in **/etc/inittab**, then it will request an initial *run-level* from the user at reboot time.

sysinit Entries of this type are executed before *init* tries to access the console (i.e., before the **Console Login:** prompt). It is expected that this entry will be used only to initialize devices on which *init* might try to ask the *run-level* question. These entries are executed and waited for before continuing.

process This is a *sh* command to be executed. The entire **process** field is prefixed with *exec* and passed to a forked *sh* as **sh -c ´exec** *command´*. For this reason, any legal *sh* syntax can appear in the *process* field. Comments can be inserted with the **;** *#comment* syntax.

Files

/etc/inittab
/etc/conf/cf.d/init.base

See Also

idmkinit(ADM), sulogin(ADM), disable(C), enable(C), getty(M), init(M), sh(C), who(C), exec(S), open(S), signal(S)

inode

format of an inode

Syntax

#include <sys/types.h>
#include <sys/ino.h>

Description

An inode for a plain file or directory in a file system has the structure defined by **<sys/ino.h>**. For the meaning of the defined types *off_t* and *time_t* see *types*(F).

Files

/usr/include/sys/ino.h

See Also

stat(S), filesystem(F), types(F)

issue

issue identification file

Description

The file **/etc/issue** contains the *issue* or project identification to be printed as a login prompt. This is an ASCII file which is read by program *getty* and then written to any terminal spawned or respawned from the *lines* file.

Files

/etc/issue

See Also

login(M)

langinfo

language information constants

Syntax

#include <langinfo.h>

Description

This is a header file that contains constants used to identify items of *nl_langinfo*(S) data. (See *nl_langinfo*(S).)

The contents of the header file are shown below.

```
/*
 *      LC_CTYPE   is not queried through this interface
 */

/*
 *      LC_NUMERIC          items:
 */

#define RADIXCHAR           2000   /* Decimal separator */
#define THOUSEP             2001   /* Thousands separator */

/*
 *      LC_TIME   items:
*/
#define D_FMT               3000   /* Date format only */
#define T_FMT               3001   /* Time format only */
#define PM_STR              3002   /* PM */
#define AM_STR              3003   /* AM */
#define D_T_FMT             3004   /* Date and time format */
#define DAY_1               3005   /* Sunday */
#define DAY_2               3006   /* Monday */
#define DAY_3               3007   /* Tuesday */
#define DAY_4               3008   /* Wednesday */
#define DAY_5               3009   /* Thursday */
#define DAY_6               3010   /* Friday */
#define DAY_7               3011   /* Saturday */
#define ABDAY_1             3012   /* Sun */
#define ABDAY_2             3013   /* Mon */
#define ABDAY_3             3014   /* Tue */
#define ABDAY_4             3015   /* Wed */
#define ABDAY_5             3016   /* Thu */
#define ABDAY_6             3017   /* Fri */
#define ABDAY_7             3018   /* Sat */
```

```
#define MON_1             3019   /* January */
#define MON_2             3020   /* February */
#define MON_3             3021   /* March */
#define MON_4             3022   /* April */
#define MON_5             3023   /* May */
#define MON_6             3024   /* June */
#define MON_7             3025   /* July */
#define MON_8             3026   /* August */
#define MON_9             3027   /* September */
#define MON_10            3028   /* October */
#define MON_11            3029   /* November */
#define MON_12            3030   /* December */
#define ABMON_1           3031   /* Jan */
#define ABMON_2           3032   /* Feb */
#define ABMON_3           3033   /* Mar */
#define ABMON_4           3034   /* Apr */
#define ABMON_5           3035   /* May */
#define ABMON_6           3036   /* Jun */
#define ABMON_7           3037   /* Jul */
#define ABMON_8           3038   /* Aug */
#define ABMON_9           3039   /* Sep */
#define ABMON_10          3040   /* Oct */
#define ABMON_11          3041   /* Nov */
#define ABMON_12          3042   /* Dec */

/*
 *  LC_COLLATE               is not queried through this interface
 */

/*
 *  LC_MESSAGES              items:
 */

#define YESSTR            5000   /* Affirmative reply to y/n question
#define NOSTR             5001   /* Negative reply to yes/no question

/*
 *  LC_MONETARY                      items:
 */

#define CRNCYSTR          6000   /* Currency symbol */
```

See Also

nl_types(F), nl_langinfo(S)

Value Added

langinfo is an extension of AT&T System V provided by the Santa Cruz Operation.

ldfcn

common object file access routines

Syntax

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

Description

The common object file access routines are a collection of functions for reading common object files and archives containing common object files. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function *ldopen*(S) allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

LDFILE *ldptr;

TYPE(ldptr) The file magic number used to distinguish between archive members and simple object files.

IOPTR(ldptr) The file pointer returned by *fopen* and used by the standard input/output functions.

OFFSET(ldptr) The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.

HEADER(ldptr) The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

(1) functions that open or close an object file

ldopen(S) and *ldaopen*[see *ldopen*(S)]
open a common object file
ldclose(S) and *ldaclose*[see *ldclose*(S)]
close a common object file

(2) functions that read header or symbol table information

ldahread(S)
read the archive header of a member of an archive file
ldfhread(S)
read the file header of a common object file
ldshread(S) and *ldnshread*[see *ldshread*(S)]
read a section header of a common object file
ldtbread(S)
read a symbol table entry of a common object file
ldgetname(S)
retrieve a symbol name from a symbol table entry or from the string table

(3) functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

ldohseek(S)
seek to the optional file header of a common object file
ldsseek(S) and *ldnsseek*[see *ldsseek*(S)]
seek to a section of a common object file
ldrseek(S) and *ldnrseek*[see *ldrseek*(S)]
seek to the relocation information for a section of a common object file
ldlseek(S) and *ldnlseek*[see *ldlseek*(S)]
seek to the line number information for a section of a common object file
ldtbseek(S)
seek to the symbol table of a common object file

(4) the function *ldtbindex*(S) which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages.

All the functions except *ldopen*(S), *ldgetname*(S), *ldtbindex*(S) return either **SUCCESS** or **FAILURE**, both constants defined in **ldfcn.h**. *Ldopen*(S) and *ldaopen*[(see *ldopen*(S)] both return pointers to an **LDFILE** structure.

Additional access to an object file is provided through a set of macros defined in **ldfcn.h**. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the

LDFILE structure into a reference to its file descriptor field.

The following macros are provided:

```
GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)
```

The STROFFSET macro calculates the address of the string table. See the manual entries for the corresponding standard input/output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access routine library **libld.a**.

See Also

fseek(S), ldahread(S), ldclose(S), ldgetname(S), ldfhread(S), ldlread(S), ldlseek(S), ldohseek(S), ldopen(S), ldrseek(S), ldlseek(S), ldshread(S), ldtbindex(S), ldtbread(S), ldtbseek(S), stdio(S), intro(M)

Notes

The macro **FSEEK** defined in the header file **ldfcn.h** translates into a call to the standard input/output function *fseek*(S). **FSEEK** should not be used to seek from the end of an archive file, since the end of an archive file may not be the same as the end of one of its object file members.

limits

file header for implementation-specific constants

Syntax

#include <limits.h>

Description

The header file **<limits.h>** is a list of magnitude limitations imposed by a specific implementation of the operating system. All values are specified in decimal.

```
#define ARG_MAX     5120       /* max length of arguments to exec */
#define CHAR_BIT    8          /* # of bits in a "char" */
#define CHAR_MAX    127        /* max integer value of a "char" */
#define CHAR_MIN    -128       /* min integer value of a "char" */
#define CHILD_MAX   25         /* max # of processes per user id */
#define CLK_TCK     100        /* # of clock ticks per second */
#define DBL_DIG     16         /* digits of precision of a "double" */
#define DBL_MAX     1.79769313486231470e+308 /*max decimal value of a "double"*/
#define DBL_MIN     4.94065645841246544e-324 /*min decimal value of a "double"*/
#define FCHR_MAX    1048576    /* max size of a file in bytes */
#define FLT_DIG     7          /* digits of precision of a "float" */
#define FLT_MAX     3.40282346638528860e+38 /*max decimal value of a "float" */
#define FLT_MIN     1.40129846432481707e-45 /*min decimal value of a "float" */
#define HUGE_VAL    3.40282346638528860e+38 /*error value returned by Math lib*/
#define INT_MAX     2147483647 /* max decimal value of an "int" */
#define INT_MIN     -2147483648 /* min decimal value of an "int" */
#define LINK_MAX    1000       /* max # of links to a single file */
#define LONG_MAX    2147483647/* max decimal value of a "long" */
#define LONG_MIN    -2147483648         /* min decimal value of a "long" */
#define Name_MAX    14         /* max # of characters in a file name */
#define OPEN_MAX    60         /* max # of files a process can have open */
#define PASS_MAX    8          /* max # of characters in a password */
#define PATH_MAX    256        /* max # of characters in a path name */
#define PID_MAX     30000      /* max value for a process ID */
#define PIPE_BUF    5120       /* max # bytes atomic in write to a pipe */
#define PIPE_MAX    5120       /* max # bytes written to a pipe in a write */
#define SHRT_MAX    32767      /* max decimal value of a "short" */
#define SHRT_MIN    -32768     /* min decimal value of a "short" */
#define STD_BLK     1024       /* # bytes in a physical I/O block */
#define SYS_NMLN    9          /* # of chars in uname-returned strings */
#define UID_MAX     60000      /* max value for a user or group ID */
#define USI_MAX     4294967295/* max decimal value of an "unsigned" */
#define WORD_BIT    32         /* # of bits in a "word" or "int" */
```

Standards Conformance

limits is conformant with:

The X/Open Portability Guide II of January 1987.

linenum

line number entries in a common object file

Syntax

#include <linenum.h>

Description

The *cc* command generates an entry in the object file for each C source line on which a breakpoint is possible [when invoked with the -g option; see *cc*(CP)]. Users can then reference line numbers when using the appropriate software test system [see *sdb*(CP)]. The structure of these line number entries appears below.

```
struct  lineno
{
        union
        {
                long      l_symndx ;
                long      l_paddr ;
        }                 l_addr ;
        unsigned short    l_lnno ;
} ;
```

Numbering starts with one for each function. The initial line number entry for a function has *l_lnno* equal to zero, and the symbol table index of the function's entry is in *l_symndx*. Otherwise, *l_lnno* is non-zero, and *l_paddr* is the physical address of the code for the referenced line. Thus the overall structure is the following:

function symtab index	0
physical address	line
physical address	line
...	
function symtab index	0
physical address	line
physical address	line
...	
l_addr	*l_lnno*

See Also

cc(CP), sdb(CP), a.out(F)

logs

MMDF log files

Syntax

System status, error, and statistics logging for MMDF

Description

MMDF maintains run-time log files at several levels of activity. The primary distinction is among message-level, channel-level, and link-level information. All logging settings can be overridden by entries in the runtime tailor file . In MMDF, that member is merged with *lusrlmmdfllog* to determine the full pathname to the log. Logs are protected so that any process may write into them, but only MMDF may read them (i.e., 0622).

The logging files may be the source of some confusion, since the llog package entails some complexity. Its three critical factors are coordinated access, restricted file length, and restricted verbosity.

The length of a logging file can be limited to 25-block units. This is extremely important since files can grow very long over a period of time, especially if there are many long messages sent or very verbose logging.

Restricted verbosity is a way of easily tuning the amount of text entered into the log. This is probably the one parameter you need most to worry about. Set to full tilt (level=FTR), MMDF get noticeably slower and I/O bound. It also does a pretty good job of showing what it is doing and hence helping you figure out the source of errors. When you get to trust the code, setting the logging level down is highly recommended. The lowest would be TMP or FAT, for temporary or fatal errors. GEN will log errors and general information. FST logs errors, general and statistics information.

Specific Logs

Even with the listed divisions, the logs contain a variety of information. Only the message-level log's format will be explained in significant detail.

msg.log records enqueue and dequeue transitions, by *submit* and *deliver*. Entries by a background *deliver* process are noted with a ''BG-xxxx'' tag, where the x's contain the 4 least-significant decimal digits of the daemon's process id. This is to allow distinguishing different daemons. When *deliver* is invoked, by *submit,* for an immediate attempt, the tag begins with ''DL'' rather than ''BG''. Entries by *submit* begin with ''SB''.

Every major entry will indicate the name of the message involved. Entries from *submit* will show ''lin'' if the submission is from a user on the local machine. In this case, the end of the entry will show the login name of the sender. If the entry is labelled ''rin,'' then the mail is being relayed. The channel name, source host, and sender address are shown. Within parentheses, the number of addressees and the byte-length of the message are listed.

Entries from *deliver* show final disposition of a message/addressee. These are indicated by ''end.'' Then, there is the destination channel and mailbox name. In brackets, the queue latency for the address is shown in hours, seconds, and minutes.

chan.log records activity by the channel programs, in chndfldir[]. Entries have a tag indicating the type of channel making the entry. Different channels record different sorts of information. For example, the *local* channel shows when a *rcvmail* private reception program is invoked.

ph.log is used by the telephone link-level (packet) code.

ph.trn is the one file that is *not* size-limited. It records a transcript of every character sent and received on a telephone channel. It is reset to zero length at the beginning of every phone session. It is kept verbose, in order to facilitate checking the status of any telephone channel which is active. Hence, just watching for the ph.trn file to get larger can indicate that there is progress. Each telephone channel may have its own transcript file specified in the channel definition in the runtime tailor file.

See Also

mmdf(ADM)

Value Added

logs is an extension of AT&T System V provided by the Santa Cruz Operation.

maildelivery

user delivery specification file

Description

The delivery of mail by the local channel can run through various courses, including using a user tailorable file. The delivery follows the following strategy, giving up at any point it considers the message delivered.

1) If the address indicates a pipe or file default, then that is carried out.

2) The file *.maildelivery* (or something similar) in the home directory is read if it exists and the actions in it are followed.

3) A system-wide file is consulted next, such as */usr/lib/maildelivery* and the actions are similar to step 2.

4) If the message still hasn't been delivered, then it is put into the user's normal mailbox (*.mail* or *mailbox*) depending on the system.

The format of the *.maildelivery* file is

field <*FS*> **pattern** <*FS*> **action** <*FS*> **result** <*FS*> **string**

where:

field is name of a field that is to be searched for a pattern. This is any header field that you might find in a message. The most commonly used headers are From, to, cc, subject and sender. As well as the standard headers, there are some pseudo-headers that can also be used. These are:

source The out-of-band sender information. This is the address MMDF would use for reporting delivery problems with the message.

addr the address that was used to mail to you, normally yourname or yourname=string (see below).

default if the message hasn't been delivered yet, this field is matched.

* this case is always true regardless of any other action.

pattern is some sequence of characters that may be matched in the above *field*. Case is not significant.

action is one of the mail delivery actions supported by the local channel. Currently the supported actions are **file** or >, which appends the message to the given file, with delimiters; **pipe** or |, which starts up a process with the message as the standard input; and **destroy** which throws the message away.
For piped commands, the exit status of the command is significant. An exit status of 0 implies that the command succeeded and everything went well. An exit status of octal 0300-0377 indicates that a permanent failure occurred and the message should be rejected. Any other exit status indicates a temporary failure and the delivery attempt will be aborted and restarted at a later time.

result is one of the letters A, R or ? which stand for Accept, Reject and "Accept if not delivered yet". They have the following effects:

A If the result of this line's action is OK, then the message can be considered delivered.

R The message is not to be considered delivered by this action.

? This is equivalent to *A* except that the action is not carried out if the message has already been accepted.

The file is always read completely so that several matches can be made, and several actions taken. As a security check, the *.maildelivery* file must be owned by either the user or root, and must not have group or general write permission. In addition the system delivery file has the above restrictions but must also be owned by root. If the field specified does not need a pattern, a dash (-), or similar symbol is usually inserted to show that the field is present but not used. The field separator character can be either a tab, space or comma (,). These characters can be included in a string by quoting them with double quotes ("); double quotes can be included with a backslash '\'.

MMDF treats local addresses which contain an equal sign ('=') in a special manner. Everything in a local address from an equals sign to the '@' is ignored and passed on to the local channel. The local channel will make the entire string available for matching against the *addr* string of the *.maildelivery* file. For example, if you were to subscribe to a digest as "foo=digest@bar.NET", *submit*(ADM) and the local channel will verify that it is legal to deliver to "foo", but then the entire string "foo=digest" will be available for string matching against the *.maildelivery* file for the **addr** field.

Environment

The environment in which piped programs are run contains a few standard features, specifically:

HOME is set to the user's home directory.
USER is set to the user's login name.
SHELL is set to the user's login shell (defaults to **/bin/sh**).

The default umask is set up to 077. This gives a very protective creation mask. If further requirements are needed, then a shell script can be run first to set up more complex environments.

There are certain built-in variables that you can give to a piped program. These are *$(sender)*, *$(address)*, *$(size)*, *$(reply-to)* and *$(info)*. *$(sender)* is set to the return address for the message. *$(address)* is set to the address that was used to mail to you, normally 'yourname' or 'yourname=string'. *$(size)* is set to the size in bytes of this message. *$(reply-to)* is set to the Reply-To: field (or the From: field if the former is missing) and so can be used for automatic replies. *$(info)* is the info field from the internal mail header and is probably only of interest to the system maintainers.

Example

Here is a rough idea of what a *.maildelivery* file looks like:

```
# lines starting with a '#' are ignored.
# as are blank lines
# file mail with mmdf2 in the "To:" line into file mmdf2.log
To    mmdf2    file    A    mmdf2.log
# Messages from mmdf pipe to the program err-message-archive
From    mmdf    pipe    A    err-message-archive
# Anything with the "Sender:" address "uk-mmdf-workers"
# file in mmdf2.log if not filed already
Sender    uk-mmdf-workers    file    ?    mmdf2.log
# "To:" unix - put in file unix-news
To    Unix    >    A    unix-news
# if the address is jpo=mmdf - pipe into mmdf-redist
Addr    jpo=mmdf    |    A    mmdf-redist
# if the address is jpo=ack - send an acknowledgement copy back
Addr    jpo=ack    |    R    resend  -r  $(reply-to)
# anything from steve - destroy!
from    steve    destroy    A    -
# anything not matched yet - put into mailbox
default    -    >    ?    mailbox
# always run rcvalert
*    -    |    R    rcvalert
```

Files

$HOME/.maildelivery normal location
/usr/lib/maildelivery the system file. This should be protected.

The **/usr/lib/maildelivery** file contains contents such as:

```
default    -    pipe    A    stdreceive
*          -    pipe    R    ttynotify
```

This allows the system to interface with non-standard mail systems that do not support delimiter-separated mailboxes.

See Also

rcvtrip(C)

mapchan

format of tty device mapping files

Description

mapchan configures the mapping of information input and output.

Each unique *channel* map requires a multiple of 1024 bytes (a 1K buffer) for mapping the input and output of characters. No buffers are required if no *channels* are mapped. If control sequences are specified, an additional 1K buffer is required.

A method of sharing maps is implemented for *channels* that have the same map in place. Each additional, unique map allocates an additional buffer. The maximum number of map buffers available on a system is configured in the kernel, and is adjustable via the link kit **NEMAP** parameter (see *configure* (ADM)). Buffers of maps no longer in use are returned for use by other maps.

Example of a Map File

The internal character set is defined by the right column of the input map, and the first column of the output map in place on that line. The default internal character set is the 8-bit ISO 8859/1 character set, which is also known as dpANS X3.4.2 and ISO/TC97/SC2. It supports the Latin alphabet and can represent most European languages.

Any character value not given is assumed to be a straight mapping, only the differences are shown in the *mapfile* . The left hand columns must be unique. More than one occurrence of any entry is an error. Right hand column characters can appear more than once. This is ‘‘many to one’’ mapping. Nulls can be produced with compose sequences or as part of an output string.

It is recommended that no mapping be enabled on the *channel* used to create or modify the mapping files. This prevents any confusion of the actual values being entered due to mapping. It is also recommended that numeric rather than character representations be used in most cases, as these are not likely to be subject to mapping. Use comments to identify the characters represented. Refer to the *ascii* (M) manual page and the hardware reference manual for the device being mapped for the values to assign.

```
#
# sharp/pound/cross-hatch is the comment character
# however, a quoted # ('#') is 0x23, not a comment
#
# beep, input, output, dead, compose and
# control are special keywords and should appear as shown.
#

    beep            # sound the bell when errors occur
    input

    a b
    c d

    dead p
    q r             # p followed by q yields r.
    s t             # p followed by s yields t.

    dead u
    v w             # u followed by v yields w.

    compose x       # x is the compose key (only one allowed).
    y z A
    B C D           # x followed by B and C yields D.

    output
    e f             # e is mapped to f.
    g h i j         # g is mapped to hij - one to many.
    k l m n o       # k is mapped to lmno.

    control         # The control sections must be last

    input
    E  1            # The character E is followed by 1 more
                    unmapped character

    output
    FG  2           # The characters FG are followed by 2
                    more unmapped characters
```

All of the single letters above preceding the ''control'' section must be in one of these formats:

56	# decimal
045	# octal
0xfa	# hexadecimal
'b'	# quoted char
'\076'	# quoted octal
'\x4a'	# quoted hex

All of the above formats are translated to single byte values.

The **control** sections (which must be the last in the file) contain specifications of character sequences which should be passed through to or from the terminal device without going through the normal *mapchan* processing. These specifications consist of two parts: a fixed sequence of one or more defined characters indicating the start of a no-map sequence, followed by a number of characters of which the actual values are unspecified.

To illustrate this, consider a cursor-control sequence which should be passed directly to the terminal without being mapped. Such a sequence would typically begin with a fixed escape sequence instructing the terminal to interpret the following two characters as a cursor position; the values of the following two characters are variable, and depend on the cursor position requested. Such a control sequence would be specified as:

```
\E=  2                          # Cursor control: escape = <x> <y>
```

There are two subsections under **control**: the **input** section is used to filter data sent from the terminal to UNIX, and the **output** section is used to filter data sent from UNIX to the terminal. The two fields in each control sequence are separated by white space, that is the SPACE or TAB characters. Also the '#' (HASH) character introduces a comment, causing the remainder of the line to be ignored. Therefore, if any of these three characters are required in the specification itself, they should be entered using one of alternative means of entering characters, as follows:

^*x* The character produced by the terminal on pressing the CONTROL and *x* keys together.

\E or \e
The ESCAPE character, octal 033.

\c Where *c* is one of b, f, l, n, r or t, produces BACKSPACE , FORM FEED , LINE FEED , NEWLINE , CARRIAGE RETURN or TAB characters respectively.

\0 Since the NULL character can not be represented, this sequence is stored as the character with octal value 0200, which behaves as a NULL on most terminals.

\nn or \nnn
Specifies the octal value of the character directly.

\ followed by any other character is interpreted as that character. This can be used to enter SPACE , TAB , or HASH characters.

Diagnostics

mapchan performs these error checks when processing the mapfile:

- More than one compose key.
- Characters mapped to more than one thing.
- Syntax errors in the byte values.
- Missing input or output keywords.
- Dead or compose keys also occurring in the input section.
- Extra information on a line.
- Mapping a character to null.
- Starting an output control sequence with a character that is already mapped.

If characters are displayed as the 7-bit value instead of the 8-bit value, use **stty -a** to verify that **-istrip** is set. Make sure **input** is mapping to the 8859 character set, **output** is mapping from the 8859 to the device display character set. **dead** and **compose** sequences are **input** mapping and should be going to 8859.

Files

/etc/default/mapchan
/usr/lib/mapchan/*

See Also

ascii(M), keyboard(HW), lp(C), lpadmin(ADM), mapchan(M), trchan(M), mapkey(M), parallel(HW), screen(HW), serial(HW), setkey(M), tty(M)

Notes

Some non-U.S. keyboards and display devices do not support characters commonly used by UNIX command shells and the C programming language. Do not attempt to use such devices for system administration tasks.

Not all terminals or printers can display all the characters that can be represented using this utility. Refer to the device's hardware manual for information on the capabilities of the peripheral device.

Warnings

Use of mapping files that specify a different ''internal'' character set per-channel, or a set other than the 8-bit ISO 8859 set supplied by default can cause strange side effects. It is especially important to retain the 7-bit ASCII portion of the character set (see *ascii*(M)). UNIX utilities and applications assume these values. Media transported between machines with different internal code set mappings may not be portable as no mapping is performed on block devices, such as tape and floppy drives. *trchan* can be used to ''translate'' from one internal character set to another.

Do not set ISTRIP (see *stty*(C)) on channels that have mapping that includes eight bit characters.

Value Added

mapchan is an extension of AT&T System V provided by the Santa Cruz Operation.

maxuuscheds

UUCP uusched(ADM) limit file

Description

The *Maxuuscheds* (*/usr/lib/uucp/Maxuuscheds*) file contains a numerical string to limit the number of simultaneous **uusched** programs running. Each **uusched** running will have one **uucico** associated with it; limiting the number will directly affect the load on the system. The limit should be less than the number of outgoing lines used by UUCP (a smaller number is often desirable). This file is delivered with a default entry of 2. Again, this may be changed to meet the needs of the local system. However, keep in mind that the load on the system increases with the number of **uusched** programs running.

See Also

uucico(ADM), uucp(C), uusched(ADM), uux(C), uuxqt(C)

maxuuxqts

UUCP uuxqt(C) limit file

Description

The *Maxuuxqts* (*/usr/lib/uucp/Maxuuxqts*) file contains an ASCII number to limit the number of simultaneous **uuxqt** programs running. This file has a default entry of 2. If there is a lot of traffic from **mail**, you can increase this number to reduce the time it takes for the mail to leave your system. Keep in mind that the load on the system increases with the number of **uuxqt** programs running.

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C)

mdevice

device driver module description file

Syntax

/etc/conf/cf.d/mdevice

Description

The *mdevice* file is included in the directory */etc/conf/cf.d*. It includes a one-line description of each device driver and configurable software module in the system to be built [except for file system types, see *mfsys*(F)]. Each line in *mdevice* represents the *Master* file component from a Driver Software Package (DSP) either delivered with the base system or installed later via *idinstall*(ADM).

Each line contains several whitespace-separated fields; they are described below. Each field must be supplied with a value or a '-' (dash).

1. *Device name*: This field is the internal name of the device or module, and may be up to 8 characters long. The first character of the name must be an alphabetic character; the others may be letters, digits, or underscores.

2. *Function list*: This field is a string of characters that identify driver functions that are present. Using one of the characters below requires the driver to have an entry point (function) of the type indicated. If no functions in the following list are supplied, the field should contain a dash.

   o - open routine

   c - close routine

   r - read routine

   w - write routine

   i - ioctl routine

   s - startup routine

   I - init routine

   h - halt routine

p - poll routine

E - enter routine

Note that if the device is a 'block' type device (see field 3. below), a *strategy* routine and a *print* routine are required by default.

3. *Characteristics of driver*: This field contains a set of characters that indicate the characteristics of the driver. If none of the characters below apply, the field should contain a dash. The legal characters for this field are:

   i - The device driver is installable.

   c - The device is a 'character' device.

   b - The device is a 'block' device.

   t - The device is a tty.

   o - This device may have only one *sdevice* entry.

   r - This device is required in all configurations of the Kernel. This option is intended for drivers delivered with the base system only. Device nodes (special files in the */dev* directory), once made for this device, are never removed. See *idmknod.*

   S - This device driver is a STREAMS module.

   H - This device driver controls hardware. This option distinguishes drivers that support hardware from those that are entirely software (pseudo-devices).

   G - This device does not use an interrupt though an interrupt is specified in the *sdevice* entry. This is used when you wish to associate a device to a specific device group.

   D - This option indicates that the device driver can share its DMA channel.

   O - This option indicates that the IOA range of this device may overlap that of another device.

4. *Handler prefix*: This field contains the character string prepended to all the externally-known handler routines associated with this driver. The string may be up to 4 characters long.

5. *Block Major number*: This field should be set to zero in a DSP *Master* file. If the device is a 'block' type device, a value will be assigned by *idinstall* during installation.

6. *Character Major number*: This field should be set to zero in a DSP *Master* file. If the device is a 'character' type device (or 'STREAMS' type), a value will be assigned by *idinstall* during installation.

7. *Minimum units*: This field is an integer specifying the minimum number of these devices that can be specified in the *sdevice* file.

8. *Maximum units*: This field specifies the maximum number of these devices that may be specified in the *sdevice* file. It contains an integer.

9. *DMA channel*: This field contains an integer that specifies the DMA channel to be used by this device. If the device does not use DMA, place a '-1' in this field. Note that more than one device can share a DMA channel (previously disallowed).

Specifying STREAMS Devices and Modules

STREAMS modules and drivers are treated in a slightly different way from other drivers in all UNIX systems, and their configuration reflects this difference. To specify a STREAMS device driver, its *mdevice* entry should contain both an 'S' and a 'c' in the *characteristics* field (see 3. above). This indicates that it is a STREAMS driver and that it requires an entry in the UNIX kernel's *cdevsw* table, where STREAMS drivers are normally configured into the system.

A STREAMS module that is not a device driver, such as a line discipline module, requires an 'S' in the *characteristics* field of its *mdevice* file entry, but should not include a 'c', as a device driver does.

See Also

mfsys(F), sdevice(F), idinstall(ADM)

mem, kmem

memory image file

Description

The *mem* file provides access to the computer's physical memory. All byte addresses in the file are interpreted as memory addresses. Thus, memory locations can be examined in the same way as individual bytes in a file. Note that accessing a nonexistent location causes an error.

The *kmem* file is the same as *mem* except that it corresponds to kernel virtual memory rather than physical memory.

In rare cases, the *mem* and *kmem* files may be used to write to memory and memory-mapped devices. Such patching is not intended for the naive user and may lead to a system crash if not conducted properly. Patching device registers is likely to lead to unexpected results if the device has read-only or write-only bits.

Files

/dev/mem

/dev/kmem

mfsys

configuration file for filesystem types

Syntax

/etc/conf/cf.d/mfsys

Description

The *mfsys* file contains configuration information for filesystem types that are to be included in the next system kernel to be built. It is included in the directory */etc/conf/cf.d*, and includes a one-line description of each filesystem type. The *mfsys* file is gathered from component files in the directory */etc/conf/mfsys.d*. Each line contains the following whitespace-separated fields:

1. *name*: This field contains the internal name for the filesystem type (example: S51K, AFS). This name is no more than 32 characters long, and by convention is composed of uppercase alphanumeric characters.

2. *prefix*: The *prefix* in this field is the string prepended to the *fstypsw* handler functions defined for this filesystem type (example: s5) The prefix must be no more that 8 characters long.

3. *flags*: The *flags* field contains a hex number of the form ‘‘0xNN’’ to be used in populating the *fsinfo* data structure table entry for this filesystem type.

4. *notify flags*: The *notify flags* field contains a hex number of the form ‘‘0xNN’’ to be used in population the *fsinfo* data structure table entry for this filesystem type.

5. *function bitstring*: The *function bitstring* is a string of 28 zeros and ones. Each filesystem type potentially defines 28 functions to populate the *fstypsw* data structure table entry for itself. All filesystem types do not supply all the functions in this table, however, and this bitstring is used to indicate which of the functions are present and which are absent. A ‘‘1’’ in this string indicates that a function has been supplied, and a ‘‘0’’ indicates that a function has not been supplied. Successive characters in the string represent successive elements of the *fstypsw* data structure, with the first entry in this data structure represented by the rightmost character in the string.

See Also

sfsys(F), idinstall(ADM), idbuild(ADM)

micnet

the Micnet default commands file

Syntax

/etc/default/micnet

Description

The **micnet** file lists the system commands that may be executed through the *remote* command. The file is required for each system in a Micnet network. Whenever a *remote* command is received through the network, the Micnet programs search the **micnet** file for the system command specified with the *remote* command. If found, the command is executed. Otherwise, the command is ignored and an error message is returned to the system which issued the *remote* command.

The file may contain one or more lines. If all commands may be executed, only the line

```
executeall
```

is required in the file. Otherwise, the commands must be listed individually. A line that defines an individual command has the form:

```
command=commandpath
```

Command is the command name to be specified in a *remote* command. **Commandpath** is the full pathname of the command on the specified system. The equal sign (=) separates the command and commandpath. For example, the line:

```
cat=/bin/cat
```

defines the command name *cat* (used in the *remote* command) to refer to the system command *cat* in the **/bin** directory.

When *executeall* is set, commands are sought in a series of default directories. Initially, the directories are **/bin** and **/usr/bin**. The default directories can be explicitly defined in the file by including a line of the form:

```
execpath=PATH=directory[:directory]...
```

The first part of the line, *execpath=PATH=*, is required. Each **directory** must be a valid pathname. The colon is required to separate directories. For example, the line:

```
execpath=PATH=/bin:/usr/bin:/usr/bobf/bin
```

sets the default directories to **/bin**, **/usr/bin**, and **/usr/bobf/bin**.

Files

/etc/default/micnet

See Also

aliases(M), netutil(ADM), systemid(F), top(F)

Notes

The **rcp** command cannot be executed from a remote system unless the **micnet** file contains either *executeall* , or the line

```
rcp=/usr/bin/rcp
```

Value Added

micnet is an extension of AT&T System V provided by the Santa Cruz Operation.

mmdftailor

provides run-time tailoring for the MMDF mail router

Description

The MMDF mail router reads site-dependent information from the ASCII file **/usr/mmdf/mmdftailor** each time it starts up.

Keywords in the tailor file are not case-sensitive; however, case is important for filenames and similar values. Use quotation marks to delimit strings to prevent them from being parsed into separate words accidentally.

The following alphabetical list describes most of the information you can set in the **mmdftailor** file. For information about additional channel-specific settings, refer to the documentation about the particular channel.

ALIAS defines an alias table. The following parameters can be used:

table specifies the name of the table to be associated with this alias entry

trusted allows the entries in the table to cause delivery to files and pipes

nobypass does not allow the ~*address* alias bypass mechanism to work on this file

Here is an example:

```
ALIAS table=sysaliases, trusted, nobypass
```

AUTHLOG controls authorization information. See MCHANLOG and MLOGDIR.

AUTHREQUEST is the address to which users should mail if they have questions about why a message was not authorized for delivery. It is also used as the sender of authorization warning messages. It is not used if authorization is not enabled on some channel. See the *auth* parameter under MCHN.

MADDID controls whether *submit* adds *Message-ID:* header lines if they are missing from messages. It takes a number as an argument. If the number is 0, no action is taken. If the number is nonzero, then *submit* adds *Message-ID:* header lines if they are missing from messages.

MADDRQ is the address files directory. If it is not a full pathname, it is taken relative to MQUEDIR.

MCHANLOG controls MMDF logging, except for authorization information and information produced by *deliver* and *submit*. See also MMSGLOG, AUTHLOG, and MLOGDIR.

Logging files and levels can also be specified in the channel descriptions. The logging file, if specified there, overrides the MCHANLOG definition. The logging level for the channel is set to the maximum of the MCHANLOG level and the channel description's level. The MCHANLOG level can therefore be used to increase logging on all channels at once.

Here is an example:

```
MCHANLOG /tmp/mmdfchan.log, level=FST, size=40,
         stat=SOME
```

An argument without an equal sign is taken as the name of the log. Logging levels are:

FAT	logs fatal errors only
TMP	logs temporary errors and fatal errors
GEN	saves the generally interesting diagnostics
BST	shows some basic statistics
FST	gives full statistics
PTR	shows a program trace listing of what is happening
BTR	shows more detailed tracing
FTR	saves every possible diagnostic

The BTR and FTR conditions are enabled only if you have compiled the MMDF system with DEBUG #define'd. This amount of tracing can quickly fill up log files. During normal operation, the logging level should be set somewhere between GEN and FST.

The *size* parameter is the number of 25-block units you will allow your log file to grow to. When a log file reaches that size, that logging either stops or cycles around overwriting old data (see **cycle**).

The *stat* parameter sets up various status flags for logging:

close closes the log after each entry; this allows other processes to write to it as well

wait if the log is busy, waits a while for it to free

cycle if the log is at the maximum length specified with the *size* parameter, then cycles to the beginning some sets the values **close** and **wait** (the most common setting)

timed opens the log and, after the timeout period (e.g., 5 minutes), closes the log and reopens it; this option overrides all other options (used to reduce the overhead of re-opening the log for every entry while still retaining the ability to move the log file out from under a running process and have the process begin logging in the new log file soon thereafter)

Tailoring of the log files is generally done at the end of the tailor file to prevent logging the tailoring action itself, thereby needlessly filling the log files when higher tracing levels are enabled. If you have bugs in the tailoring, you can move the log-file tailoring closer to the top of the tailor file.

MCHN defines a channel. The following parameters can be used:

name the name of the channel

show a display line, which is used for pretty printing purposes to explain what the channel is all about

que the queue directory into which messages for this channel should be queued

tbl the associated table entry of hosts that are accessible on this channel; if the specified table has not been previously defined, it will be defined with the same *name*, *file*, and *show* parameters as for this channel (do not define two channels that process the same queue, but use different tables because it will cause queue structure problems)

pgm the channel program to invoke for this channel

mod the mode in which this channel works; if several values are selected, they are cumulative:

> reg regular mode (the default); the channel must be explicitly invoked by running *deliver*
>
> host same as **reg**, but causes *deliver* to sort by host after sorting by channel, which allows as many mail messages as possible to get sent to a particular host before the connection is broken

bak channel can be invoked only by the background deliver daemon

psv channel is passive; it is a pickup-type channel (e.g., pobox)

imm channel can be invoked immediately; no need to batch up mail

pick channel can pick up mail from the remote host

send channel can send mail to the remote host

ap the address-parser type to be used for reformatting headers on messages going out on this channel; if several values are selected, they are cumulative:

same does not reformat headers

822 selects RFC822-style source routes (e.g., @A:B@C)

733 selects %-style (JNT) source routes (e.g., B%C@A)

big selects NRS hierarchy ordering (for the UK)

nodots selects output of leftmost part of domain names (e.g., A in A.B.C) for sites that cannot handle domains (usually used in conjunction with the *known*= parameter to hide domain names behind a smart relay)

jnt is equivalent to **733** combined with **big**

lname a name overriding the default MLNAME value for this channel (used when the channel should have non-standard values for the local domain)

ldomain a name overriding the default MLDOMAIN value for this channel host the name of the host that is being contacted by this channel, usually used in the phone and pobox channels, or the name of the relay host when all mail to hosts in this channel's table gets relayed to one host (this is required on the *badusers* and *badhosts* pseudo-channels; it must be set to the local host for the list channel)

poll the frequency of polling the remote machine (0 disables polling, -1 requests polling to be done every time the channel is started up, any other value is the number of 15-minute intervals to wait between polls); if any mail is waiting to be sent, this value is ignored because a connection is always attempted

insrc a table of hosts controlling message flow

outsrc see *insrc*

indest see *insrc*

outdest see *insrc*

known a table of hosts that are known on this channel; be sure that the table contains your own fully specified host name

confstr a string used by some channels for specifying the invocation of another program

auth specifies the authorization tests that are made on this channel:

- free default, no checking takes place
- inlog log information for incoming messages
- outlog log information for outgoing messages
- inwarn warn sender of incoming message if authorization is inadequate (the message still goes through)
- outwarn as **inwarn**, but for outgoing messages
- inblock reject incoming messages that have inadequate authorization

outblock as **inblock**, but for outgoing messages

hau host and user authorizations are required

dho (direct host only) when applying host controls, the message must be associated with a user local to that host (i.e., no source routes)

ttl (time-to-live) specifies the number of minutes for which retries to a host are blocked when *deliver* detects a connection failure to that host; this value can be overridden on the *deliver* command line (default is 2 hours)

log the name of the channel log file to be used instead of the default MCHANLOG

level the logging level for this channel (see also MCHANLOG)

Here is a simple example:

```
MCHN name=local, que=local, tbl=local,
     show="Local Delivery", pgm=local,
     poll=0, mod=imm, ap=822, level=BST
```

If the first argument does not have an equal sign, the values of the *name*, *que*, *tbl*, *pgm*, and *show* parameters take on this value.

MCHNDIR is where the channel programs are to be found.

MCMDDIR is the default commands directory where the various MMDF commands are located. Any command that does not have a full pathname is taken relative to this directory.

MDBM tells MMDF where to find the database file containing the associative store. DBM-style databases get their speed and flexibility by dynamic hashing and can get quite large. By default, the file is located in the MTBLDIR directory, but it might need to be relocated due to its size.

MDFLCHAN sets the default channel to something other than local.

MDLV is the name of the file used for tailoring the delivery for each user.

MDLVRDIR is the directory in which to deliver mail. If this is null, then the user's home directory is used. See also MMBXNAME and MMBXPROT.

MDMN defines a domain. The following parameters can be used:

name an abbreviated name for the domain

show a display line, which is used for pretty printing purposes to explain what the domain is all about

dmn the full name (x.y.z...) of this domain

table the associated table entry of known sites in this domain; if the specified table has not been previously defined, it will be defined with the same *name*, *file*, and *show* parameters as for this domain

Here is an example:

```
MDMN name="Root", dmn="", show="Root Domain",
    table=rootdomain
```

If the first argument does not have an equal sign, the values of the *name*, *dmn*, and *show* parameters take on this value. If no *table* parameter is specified, it defaults to the value of the name parameter.

MFAILTIME is the time a message can remain in a queue before a failed-mail message is sent to the sender and the message is purged from the queue. See also MWARNTIME.

MLCKDIR is the directory where the locking of files takes place, this is dependent on what style of locking you are doing.

MLDOMAIN gives your full local domain (this, combined with the MLNAME, and possibly the MLOCMACHINE, gives the full network address).

MLISTSIZE specifies the maximum number of addresses in a message before it is considered to have a ''big'' list. If there are more than the maximum number of addresses, then MMDF does not send a warning message for waiting mail and only returns a ''citation'' for failed mail. A citation consists of the entire header plus the first few lines of the message body.

MLNAME is the name of your machine or site as you wish it to be known throughout the network, which can be a generic host name used to hide a number of local hosts. If it is a generic host name, internal hosts are differentiated by MLOCMACHINE. See also MLDOMAIN.

MLOCMACHINE is the local name of the machine. It is used by a site that has several machines, but wants the machines themselves to appear as one address. See also MLNAME and MLDOMAIN.

MLOGDIR is the default directory in which the log files are kept. See also MMSGLOG, AUTHLOG, and MCHANLOG.

MLOGIN is the user who owns the MMDF transport system.

MMAXHOPS specifies the maximum number of *Received:* or *Via:* lines in a message before it is considered to be looping and is rejected.

MMAXSORT controls sorting of messages based on the number of messages in the queue. If the number of messages in the queue is less then MMAXSORT, the messages are sorted by arrival time and are dispatched in that order; otherwise, a message is dispatched as it is found during the directory search.

MMBXNAME is the name of the mailbox. If this is null, then the user's login name is used. See also MDLVRDIR and MMBXPROT.

MMBXPREF is a string written before the message is written into the mailbox. It is usually set to a sequence of CTRL-A characters. See also MMBXSUFF.

MMBXPROT gives the protection mode in octal for the user's mailbox. See also MDLVRDIR and MMBXNAME.

MMBXSUFF is a string written after the message is written into the mailbox. It is usually set to a sequence of CTRL-A characters. See also MMBXPREF.

MMSGLOG controls the logging information produced by *deliver* and *submit*. See also MCHANLOG, AUTHLOG, and MLOGDIR.

MMSGQ is the directory for the files of message text. If it is not a full pathname, it is taken relative to MQUEDIR.

MPHSDIR is the directory in which the timestamps for the channels are made, showing what phase of activity they are in.

MQUEDIR is the parent directory for the various queues and address directories.

MQUEPROT gives the protection mode in octal that the parent of the MQUEDIR directory should have.

MSIG is the signature that MMDF uses when notifying senders of mail delivery problems.

MSLEEP is the length of time in seconds that the background delivery daemon sleeps between queue scans.

MSUPPORT is the address to which to send mail that MMDF cannot cope with (i.e., that MMDF cannot deliver or return to its sender).

MTBL defines an alias, domain, or channel table. The following parameters can be used:

name	a short name by which the table can be referred to later in the file
file	the file from which the contents of the table are built
show	a display line, which is used for pretty printing purposes to explain what the table is all about

A typical example might be:

```
MTBL name=aliases, file=aliases,
     show="User & list aliases"
```

If the first argument does not have an equal sign, the values of the other parameters take on this value. The following example sets the *name*, *file*, and *show* parameters to the string "aliases", then resets the *show* parameter to the string "Alias table".

```
MTBL aliases, show="Alias table"
```

MTBLDIR is the default directory for the table files.

MTEMPT is the temporary files directory. If it is not a full pathname, it is taken relative to MQUEDIR.

MWARNTIME specifies the time in hours that a message can remain in a queue before a warning message about delayed delivery is sent to the sender. See also MFAILTIME.

UUname defines the UUCP sitename (short form, not full path) and is used only by the UUCP channel. See also MLNAME.

See Also

dbmbuild(ADM), mmdf(ADM), tables(F), "Setting Up Electronic Mail" in the *System Administrator's Guide*

Value Added

mmdftailor is an extension of AT&T System V provided by the Santa Cruz Operation.

mnttab

format of mounted filesystem table

Syntax

#include <stdio.h>
#include <mnttab.h>

Description

The **/etc/mnttab** file contains a table of devices mounted by the *mount*(ADM) command.

Each table entry contains the pathname of the directory on which the device is mounted, the name of the device special file, the read/write permissions of the special file, and the date on which the device was mounted.

The maximum number of entries in *mnttab* is based on the system parameter NMOUNT, which defines the number of allowable mounted special files.

See Also

mount(ADM)

mtune

tunable parameter file

Syntax

/etc/conf/cf.d/mtune

Description

The *mtune* file contains information about all the system tunable parameters. Each tunable parameter is specified by a single line in the file, and each line contains the following whitespace-separated set of fields:

1. *parameter name*: A character string no more than 20 characters long. It is used to construct the preprocessor ‘‘#define’s’’ that pass the value to the system when it is built.

2. *default value*: This is the default value of the tunable parameter. If the value is not specified in the *stune* file, this value will be used when the system is built.

3. *minimum value*: This is the minimum allowable value for the tunable parameter. If the parameter is set in the *stune* file, the configuration tools will verify that the new value is equal to or greater than this value.

4. *maximum value*: This is the maximum allowable value for the tunable parameter. If the parameter is set in the *stune* file, the configuration tools will check that the new value is equal to or less than this value.

The file *mtune* normally resides in */etc/conf/cf.d*. However, a user or an add-on package should never directly edit the *mtune* file to change the setting of a system tunable parameter. Instead the *idtune* command should be used to modify or append the tunable parameter to the *stune* file.

In order for the new values to become effective, the UNIX system kernel must be rebuilt and the system must then be rebooted.

See Also

stune(F), idbuild(ADM), idtune(ADM)

mvdevice

video driver backend configuration file

Syntax

/etc/conf/cf.d/mvdevice

Description

The *mvdevice* file accomplishes configurability of video hardware by permitting the linking of back ends to the console video driver. This linking scheme includes a C library of video back ends for use with the link kit and separate driver entries for each of the back ends.

The configuration program uses the *mvdevice* file to produce a **space.c** for the console driver. This **space.c** includes the appropriate include files and extern references to the appropriate video back ends. In addition, the configuration program builds the console display switch with in the **space.c**.

The *mvdevice* file has the following entries:

prefix
: name of driver from 1 to 4 characters long (for example ''mono'') This is the name of the video back end.

routine_mask
: This mask tells which routines were supported by the particular back end. These routines are: vvvinit(), vvvcmos(), vvvinitscreen(), vvvscroll(), vvvcopy(), vvvclear(), vvvpchar(), vvvscurs(), vvvsgr(), vvvioctl(), vvvadapctl().

type
: This is placed in the file as a literal. for example, if the word MONO was put into the file, it would include the word MONO as the type entry of the adapter structure.

oem
: OEM information treated exactly the same as the type (i.e. a literal)

paddr
: the physical address which the video RAM is located. This would allow a user to configure a future driver. Also included as a literal field.

size
: The size of the video RAM. Also

included as a literal field.

This information provides all the basic information needed for the program to generate an appropriate **space.c** and build the the correct adapter switch.

The routine mask uses the following bits to signify the following routines:

0x0001	vvvinit()
0x0002	vvvcmos()
0x0004	vvvinitscreen()
0x0008	vvvscroll()
0x0010	vvvcopy()
0x0020	vvvclear()
0x0040	vvvpchar()
0x0080	vvvscurs()
0x0100	vvvsgr()
0x0200	vvvioctl()
0x0400	vvvadapctl()

The default mvdevice file looks like this:

```
#
#       mvdevice:       video configuration master file.
#
#
#prefix name    routinestype    oem     paddr   size
mono    MONO    0x07fd          MONO    0       0       0
cga     CGA     0x07fd          CGA     0       0       0
ega     EGA     0x07ff          EGA     0       0       0
vga     VGA     0x07ff          VGA     0       0       0
```

See Also

sdevice(F)

nl_types

data types for native language support

Syntax

#include <nl_types.h>

Description

This is a header file that provides type definitions used in the native language support interface. The types are:

nl_item
A type defintion for an item of language data as used by *nl_langinfo*(S). The values for **nl_item** are defined in the file **<langinfo.h>**.

nl_catd
A message catalogue descriptor. (Message catalogues are not currently supported.)

See Also

langinfo(F), nl_langinfo(S)

Value Added

nl_types is an extension of AT&T System V provided by the Santa Cruz Operation.

null

the null file

Description

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

Files

/dev/null

Standards Conformance

null is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
and The X/Open Portability Guide II of January 1987.

passwd

the password file

Description

passwd contains the following information for each user:

-Login name

-Numerical user ID

-Numerical group ID

-Comment

-Initial working directory

-Program to use as shell

Refer to *finger*(C) for information in the required format of the comment field for *finger*(C) to display the information. Each user is separated from the next by a newline. If the password field is null, no password is demanded; if the shell field is null, *sh*(C) is used.

This file resides in the directory **/etc**. Encrypted passwords are not stored in **/etc/passwd**.

Warning

Under no circumstances should you edit **/etc/passwd** with a text editor. This will cause a series of error messages to be displayed and could prevent any further logins. Use the *sysadmsh* Accounts selection to modify or add user accounts.

Files

/etc/passwd

See Also

login(M), passwd(C), a64l(S), getpwent(S), group(F)

Standards Conformance

acct is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
The X/Open Portability Guide II of January 1987;
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

permissions

format of UUCP Permissions file

Description

The **Permissions** file (**/usr/lib/uucp/Permissions**) specifies the permissions for remote computers concerning login, file access, and command execution. In the **Permissions** file, you can specify the commands that a remote computer can execute and restrict its ability to request or receive files queued by the local site.

Each entry is a logical line with physical lines terminated by a \ to indicate continuation. Entries are made up of options delimited by white space. Each option is a name-value pair in the following format:

name=value

Note that no white space is allowed within an option assignment.

Comment lines begin with a pound sign (#) and they occupy the entire line up to a newline character. Blank lines are ignored (even within multi-line entries).

There are two types of **Permissions** file entries:

LOGNAME specifies the permissions that take effect when a remote computer calls your computer.

MACHINE specifies permissions that take effect when your computer calls a remote computer.

Examples

This entry is for public login. It provides the default permissions. Note that use of this type of anonymous login is not encouraged.

```
LOGNAME=nuucp \
MACHINE=OTHER \
READ=/usr/spool/uucppublic \
WRITE=/usr/spool/uucppublic \
SENDFILES=call REQUEST=no \
COMMANDS=/bin/rmail
```

See Also

uucico(ADM), uucp(C), uux(C), uuxqt(C)

plot

graphics interface

Description

Files of this format are produced by routines described in *plot*(S) and are interpreted for various devices by commands described in *tplot*(ADM). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the **x** and **y** values; each value is a signed integer. The last designated point in an **l, m, n,** or **p** instruction becomes the ''current point'' for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *plot*(S).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes [see *tplot*(ADM)].

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a new-line.

e erase: Start another frame of output.

f linemod: Take the following string, up to a new-line, as the style for drawing further lines. The styles are ''dotted'', ''solid'', ''longdashed'', ''shortdashed'', and ''dotdashed''. Effective only for the **-T4014** and **-Tver** options of *tplot*(ADM) (TEKTRONIX 4014 terminal and VERSATEC plotter).

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *tplot*(ADM). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);
DASI 450	space(0, 0, 4096, 4096);
TEKTRONIX 4014	space(0, 0, 3120, 3120);
VERSATEC plotter	space(0, 0, 2048, 2048);

See Also

plot(S), term(M), graph(ADM), tplot(ADM)

Notes

The plotting library *plot*(S) and the curses library *curses*(S) both use the names erase() and move(). The curses versions are macros. If you need both libraries, put the *plot*(S) code in a different source file than the *curses*(S) code, and/or #undef move() and erase() in the *plot*(S) code.

pnch

file format for card images

Description

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0-80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

poll

format of UUCP Poll file

Description

The **Poll** file (**/usr/lib/uucp/Poll**) contains information for polling remote computers. Each entry in the **Poll** file contains the name of a remote computer to call, followed by a tab character, and the hours the computer should be called. The hours must be integers in the range 0-23.

Poll file entries have the following format:

sysname<TAB>*hour* ...

The following entry provides polling of computer *gorgon* every four hours:

```
gorgon  0 4 8 12 16 20
```

The **uudemon.poll** script controls polling but does not perform the poll. It sets up a polling file (**C.sysnxxxx**) in the /usr/spool/uucp/*nodename* directory, where *nodename* is replaced by the name of the machine. This file will in turn be acted upon by the scheduler (started by **uudemon.hour**). The **uudemon.poll** script is scheduled to run semi-hourly before **uudemon.hour** so that the work files will be there when **uudemon.hour** is called. The default root **crontab** entry for **uudemon.poll** is as follows:

```
1,30 * * * * "/usr/lib/uucp/uudemon.poll > /dev/null"
```

See Also

uucico(ADM), uucp(C), cron(C), crontab(C)

Standards Conformance

poll is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

purge

the policy file of the sanitization utility purge(C)

Syntax

/etc/default/purge

Description

This file is an ascii file whose lines each designate a file, filesystem or device to be a member of a *type*. The command:

purge -t *type*

would overwrite all the members of *type*.

Blank lines and lines beginning with '#' are ignored. Entries are of the form:

```
file          type          [count]
```

This specifies that *file* is a member of *type*. The optional field *count* is the number of times to overwrite *file* when it is purged. The default is one.

The two *types* **system** and **user** are hardwired into the *purge*(C) utility. These types can be overwritten with the **-s** and **-u** switches to *purge* respectively.

This file should be configured on site to reflect files and devices which are sensitive and need to be protected from unauthorized access.

The initial contents of the file is:

```
/tmp          system
/user         user
```

Files

/etc/default/purge

See Also

purge(C), sysadmsh(ADM)

Value Added

purge is an extension of AT&T System V provided by the Santa Cruz Operation.

queue

MMDF queue files for storing mail in transit

Syntax

/usr/spool/mmdf/lock/home

Description

MMDF stores mail in an isolated part of the file system, so that only authorized software may access the mail. Mail is stored under the directory sub-tree.

It must specify a path with at least two sub-directories. The next-to-bottom one is used as a ''locking'' directory and the bottom one is the **home**. For full protection, only authorized software can move through the locking directory. Hence, it is owned by MMDF and accessible only to it.

Queue Entries

When mail is queued by *submit,* it is actually stored as two files. One contains the actual message text and the other contains some control information and the list of addressees.

The text file is stored in the **msg** directory. The other file is stored in the **addr** directory and is linked into one or more queue directories. The queue directories are selected based on the channels on which this message will be delivered. Each output channel typically has its own queue directory.

Another directory below **home** is a temporary area called **tmp**. It holds temporary address-lists as they are being built. Queuing of a message is completed by linking this address file into **addr** and the queue directories. The **msg** directory contains files with message text. **Addr** and **msg** files are synchronized by giving them the same unique name, which MMDF occasionally calls the message ''name''. The message name is derived by use of *mktemp(S),* using **msg** as the base directory. The files created in **addr** must be open read-write access; the ones in **msg** must be open read access.

When *submit* runs, it changes into **home** for its working directory. It then does a *setuid*() to run as the caller. This is necessary to permit *submit* to access the caller's address-list files (specified as ''< file''), but probably will be changed. *Deliver* changes into the queue directory to minimize the time spent searching for messages to deliver.

The following depicts the directory organization:

```
                        lock              (lock: 0700)
                         |                (mmdf only)
                         |
              _________home______         (open: 0777)
             /     |        |     \
            /      |        |      \
        tmp      addr      q.*     msg    (open: 0777)

     addresses  ==> moved    and linked   message text
     built here ==> into here  into here  put here

     entries:       0666       0666       0644
```

Queue File Formats

The **msg** portion of a queued message simply contains the message, which must conform to the Arpanet standard syntax, specified in RFC822. It is expected that the format of the message contents file eventually will be more structured, permitting storage of multi-media mail.

The following specifies the syntax of the **addr** (and queue directory) address-list portion of the queued message:

Address File Contents

file ::= creation late flags '\n' [rtrn-addr] '\n' *(adr_queue '\n')

creation ::= {long integer decimal representation of time message was created}

late ::= ADR_MAIL / ADR_DONE {from adr_queue.h}

flags ::= {decimal representation of 16-bits of flags}

rtrn-addr ::= {rfc822 return address}

adr_queue ::= temp_ok mode queue host local {conforms to structure specified in adr_queue.h}

temp_ok ::= {temporary flag indicating whether this address has been verified by the receiving host: ''yes'' is ''+''; ''not yet'' is ''-''}

mode ::= {send to mailbox(m), tty(t), both(b), either(e), or processing completed(*)}

queue ::= {name of the queue into which this message is linked for this address}

host ::= {official name (and domain) of recipient host}

local ::= {local address on receiving host}

Address File Description

An address queue file contains a creation time-stamp, an indication if the sender has been notified of delayed delivery, some flags, an optional return-mail address, and an address list. Several <flags> are currently in use (as specified in msg.h). ADR_NOWARN indicates whether late warnings should be sent to the return-mail address if the entire address list has not been processed within the number of hours specified by ``warntime''. ADR_NORET indicates whether mail should be returned to the sender if it hasn't been completely processed within the number of hours specified by ``failtime''. ADR_CITE indicates whether warning and failure messages are to contain only a citation of the message. An ADR_DONE (``*'') value, for the ``late'' flag, indicates that a warning notice has been sent.

The creation date is coded as a long ASCII decimal string, terminated by the ``late'' flag. This has to be stuffed into the file because Unix doesn't maintain it. The date is used to sort the queue so that mail is delivered in the order submitted.

The return address is a line of text and may be any address acceptable to *submit.*

Each address entry is on a separate line, and conforms to the adr_struct format, defined in *adr_queue.h*. It contains several fields separated by spaces or commas. Fields containing spaces or commas must be enclosed in double quotes.

The temp_ok flag indicates whether the address has been accepted during an ``address verification'' dialog with the receiving host. When the message text is successfully accepted by the receiving host, then this flag no longer applies and the mode is set to ADR_DONE (``*''). Before final acceptance of message text, the mode flag indicates whether the mail is for a mailbox, terminal, both, or either. (Currently only mailbox delivery, ADR_MAIL, is used.)

The queue name is the name of the sub-queue in which the message is queued for this address. Each addressee's host may be on a separate queue or some hosts may share the same queue. When all addressees in the same queue have been delivered, the address file is removed from that queue directory. When all queues have been processed, the address file (in both **addr** and the queue directory) and the text file (in **msg**) are removed.

The host and local strings are used by the channel program. The host determines the type of connection the channel program makes. The local string is passed to the host; it should be something meaningful to that host. The local string must not contain newline or null and it be a valid local address per RFC822.

See Also

deliver(ADM), cleanque(ADM), submit(ADM)

queuedefs

scheduling information for cron queues

Description

The *queuedefs* file is read by the clock daemon, *cron*, and controls how jobs submitted with *at*, *batch*, and *crontab* are executed. Every job submitted by one of these programs is placed in a certain queue, and the behavior of these queues is defined in **/usr/lib/cron/queuedefs**. Queues are designated by a single, lowercase letter. The following queues have special significance:

a	*at* queue
b	*batch* queue
c	*cron* queue

For a given queue, the *queuedefs* file specifies the maximum number of jobs that may be executing at one time (*njobs*), the priority at which jobs will execute (*nice*), and the how long *cron* will wait between attempts to run a job (*wait*). If *njobs* jobs are already running in a given queue when a new job is scheduled to begin execution, *cron* will reschedule the job to execute *wait* seconds later. A typical file might look like this:

```
a.4j1n
b.2j2n90w
```

Each line gives parameters for one queue. The line must begin with a letter designating a queue, followed by a period (.). This is followed by the numeric values for *njobs*, *nice*, and *wait*, followed respectively by the letters "j", "n", and "w". The values must appear in this order, although a value and its corresponding letter may be omitted entirely, in which case a default value is used. The default values are *njobs* = 100, *nice* = 2, and *wait* = 60.

The value for *nice* is added to the default priority of the job (a higher numerical priority results in a lower scheduling priority - see *nice*(C)). *wait* is given in seconds.

Files

/usr/lib/cron/queuedefs	*queuedefs* file

reloc

relocation information for a common object file

Syntax

#include <reloc.h>

Description

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format.

```
struct   reloc
{
         long        r_vaddr ;    /* (virtual) address of
                                  reference */
         long        r_symndx ;   /* index into symbol table */
         short       r_type ;     /* relocation type */
} ;

# define R_PCRLONG  024
```

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated.

R_PCRLONG A ''PC-relative'' 32-bit reference to the symbol's virtual address.

More relocation types exist for other processors. Equivalent relocation types on different processors have equal values and meanings. New relocation types will be defined (with new values) as they are needed.

Relocation entries are generated automatically by the assembler and automatically used by the link editor. Link editor options exist for both preserving and removing the relocation entries from object files.

See Also

as(CP), ld(CP), a.out(F), syms(F)

sccsfile

format of an SCCS file

Description

An SCCS file is an ASCII file. It consists of six logical parts: the *checksum*, the *delta table* (contains information about each delta), *user names* (contains login names and/or numerical group IDs of users who may add deltas), *flags* (contains definitions of internal keywords), *comments* (contains arbitrary descriptive information about the file), and the *body* (contains the actual text lines intermixed with control lines). Each logical part of an SCCS file is described in detail below.

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as *the control character* and will be represented graphically as @. Any line described below which is not depicted as beginning with the control character is prevented from beginning with the control character. Entries of the form DDDDD represent a five digit string (a number between 00000 and 99999).

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

@hDDDDD

The value of the checksum is the sum of all characters, except those of the first line. The @hR provides a *magic number* of (octal) 064001.

Delta Table

The delta table consists of a variable number of entries of the form:

@s DDDDD/DDDDD/DDDDD
@d <type> **<SCCS ID>** yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@**i** DDDDD ...
@**x** DDDDD ...
@**g** DDDDD ...
@**m** <MR number>
.
.
@**c** <comments> ...
.
.
@**e**

The first line (@s) contains the number of lines inserted/deleted/unchanged respectively. The second line (@d) contains the type of the delta (currently, normal: D, and removed: R), the SCCS ID of the delta, the date and time of creation of the delta, the login name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The @i, @x, and @g lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The @m lines (optional) each contain one MR number associated with the delta; the @c lines contain comments associated with the delta.

The @e line ends the delta table entry.

User Names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines @u and @U. An empty list allows anyone to make a delta.

Flags

Keywords used internally (see *admin*(CP) for more information on their use). Each flag line takes the form:

@f <flag>   <optional text>

The following flags are defined:

@f t	<type of program>
@f v	<program name>
@f i	
@f b	
@f m	<module name>
@f f	<floor>
@f c	<ceiling>
@f d	<default-sid>
@f n	
@f j	
@f l	<lock-releases>
@f q	<user defined>

The **t** flag defines the replacement for the identification keyword. The **v** flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The **i** flag controls the warning/error aspect of the ''No id keywords'' message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message will cause a ''fatal'' error (the file will not be gotten, or the delta will not be made). When the **b** flag is present the **-b** option may be used with the *get* command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the sccsfile.F identification keyword. The **f** flag defines the ''floor'' release; the release below which no deltas may be added. The **c** flag defines the ''ceiling'' release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a *get* command. The **n** flag causes *delta* to insert a ''null'' delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (e.g., when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes *get* to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are *locked* against editing (*get*(CP) with the **-e** option). The **q** flag defines the replacement for the identification keyword.

Comments

Arbitrary text surrounded by the bracketing lines @t and @T. The comments section typically contains a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines don't begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, as follows:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

The digit string (DDDDD) is the serial number corresponding to the delta for the control line.

See Also

admin(CP), delta(CP), get(CP), prs(CP)

Standards Conformance

sccsfile is conformant with:

AT&T SVID Issue 2, Select Code 307-127.

scnhdr

section header for a common object file

Syntax

#include <scnhdr.h>

Description

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct  scnhdr
{
        char            s_name[SYMNMLEN]; /* section name */
        long            s_paddr;     /* physical address */
        long            s_vaddr;     /* virtual address */
        long            s_size;      /* section size */
        long            s_scnptr;    /* file ptr to raw data */
        long            s_relptr;    /* file ptr to relocation */
        long            s_lnnoptr;   /* file ptr to line numbers */
        unsigned short  s_nreloc;    /* # reloc entries */
        unsigned short  s_nlnno;     /* # line number entries */
        long            s_flags;     /* flags */
} ;
```

File pointers are byte offsets into the file; they can be used as the offset in a call to FSEEK [see *ldfcn*(F)]. If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. It has a size, symbols defined in it, and symbols that refer to it. But it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for *s_scnptr*, *s_relptr*, *s_lnnoptr*, *s_nreloc*, and *s_nlnno* are zero.

See Also

ld(CP), fseek(S), a.out(F)

scr_dump

format of curses screen image file

Syntax

scr_dump(file)

Description

The *curses*(S) function *scr_dump*() will copy the contents of the screen into a file. The format of the screen image is as described below.

The name of the tty is 20 characters long and the modification time (the *mtime* of the tty that this is an image of) is of the type *time_t*. All other numbers and characters are stored as *chtype* (see **<curses.h>**). No new-lines are stored between fields.

```
<magic number:  octal 0433>
<name of tty>
<mod time of tty>
<columns> <lines>
<line length> <chars in line>for each line on the screen
<line length> <chars in line>
  .
  .
  .
<labels?>                        1, if soft screen labels are present
<cursor row> <cursor column>
```

Only as many characters as are in a line will be listed. For example, if the *<line length>* is **0**, there will be no characters following *<line length>*. If *<labels?>* is TRUE, following it will be

```
<number of labels>
<label width>
<chars in label 1>
<chars in label 2>
   .
   .
   .
```

See Also

curses(S)

sdevice

local device configuration file

Syntax

/etc/conf/cf.d/sdevice

Description

The *sdevice* file contains local system configuration information for each of the devices specified in the *mdevice* file. It contains one or more entries for each device specified in *mdevice*. *sdevice* is present in the directory */etc/conf/cf.d*, and is coalesced from component files in the directory */etc/conf/sdevice.d*. Files in */etc/conf/sdevice.d* are the *System* file components either delivered with the base system or installed later via *idinstall*.

Each entry must contain the following whitespace-separated fields:

1. *Device name:* This field contains the internal name of the driver. This must match one of the names in the first field of an *mdevice* file entry.

2. *Configure:* This field must contain the character 'Y' indicating that the device is to be installed in the kernel. For testing purposes, an 'N' may be entered indicating that the device will not be installed.

3. *Unit:* This field can be encoded with a device dependent numeric value. It is usually used to represent the number of subdevices on a controller or pseudo-device. Its value must be within the minimum and maximum values specified in fields 7 and 8 of the *mdevice* entry.

4. *Ipl*: The *ipl* field specifies the system ipl level at which the driver's interrupt handler will run in the new system kernel. Legal values are 0 through 8. If the driver doesn't have an interrupt handling routine, put a 0 in this field.

5. *Type:* This field indicates the type of interrupt scheme required by the device. The permissible values are:

   0 - The device does not require an interrupt line.

   1 - The device requires an interrupt line.
   If the driver supports more than one hardware controller, each controller requires a separate interrupt.

2 - The device requires an interrupt line.
If the driver supports more than one hardware controller, each controller will share the same interrupt.

3 - The device requires an interrupt line.
If the driver supports more than one hardware controller, each controller will share the same interrupt. Multiple device drivers having the same ipl level can share this interrupt.

6. *Vector:* This field contains the interrupt vector number used by the device. If the *Type* field contains a 0 (i.e., no interrupt required), this field should be encoded with a 0. Note that more than one device can share an interrupt number.

7. *SIOA:* The *SIOA* field (Start I/O Address) contains the starting address on the I/O bus through which the device communicates. This field must be within 0x1 and 0x3fff. (If this field is not used, it should be encoded with the value zero.)

8. *EIOA:* The field (End I/O Address) contains the end address on the I/O bus through which the device communicates. This field must be within 0x1 and 0x3fff. (If this field is not used, it should be encoded with the value zero.)

9. *SCMA:* The *SCMA* field (Start Controller Memory Address) is used by controllers that have internal memory. It specifies the starting address of this memory. This field must be within 0xa0000 and 0xfbfff. (If this field is not used, it should be encoded with the value zero.)

10. *ECMA:* The *ECMA* (End Controller Memory Address) specifies the end of the internal memory for the device. This field must be within 0xa0000 and 0xfbfff. (If this field is not used, it should be encoded with the value zero.)

See Also

mdevice(F), idinstall(ADM)

sfsys

local filesystem type file

Syntax

/etc/conf/cf.d/sfsys

Description

The *sfsys* file contains local system information about each file system type specified in the *mfsys* file. It is present in the directory */etc/conf/cf.d*, and contains a one-line entry for each file system type specified in the *mfsys* file. The *sfsys* file is coalesced from component files in the directory */etc/conf/sfsys.d*. Each line in this file is a whitespace-separate set of fields that specify:

1. *name*: This field contains the internal name of the file system type (e.g., DUFST, S51K). By convention, this name is up to 32 characters long, and is composed of all uppercase alphanumeric characters.

2. *Y/N*: This field contains either an uppercase ''Y'' (for ''yes'') or an uppercase ''N'' (for ''no'') to indicate whether the named file system type is to be configured into the next system kernel to be built.

See Also

mfsys(F), idinstall(ADM), idbuild(ADM)

stat

data returned by stat system call

Syntax

#include <sys/stat.h>

Description

The **sys/stat.h** include file contains the definition for the structure returned by the *stat* and *fstat* functions. The structure is defined as:

```
struct stat{
   dev_t       st_dev;     /*


   ino_t       st_ino;     /* inode number */
   ushort      sh_mode;    /* file mode */
   short       st_nlink;   /* # of links */
   ushort      st_uid;     /* owner uid */
   ushort      st_gid;     /* owner gid */
   dev_t       st_rdev;    /*


   off_t       st_size;    /* file size in bytes */
   time_t      st_atime;   /* time of last access */
   time_t      st_mtime;   /* time of last data modification */
   time_t      st_ctime;   /* time of last file status 'change' */
};
```

Note that the *st_atime*, *st_mtime*, and *st_ctime* values are measured in seconds since 00:00:00 (GMT) on January 1, 1970.

The *st_mode* value is actually a combination of one or more of the following file mode values:

```
S_IFMT      0170000    /* type of file */
S_IFDIR     0040000    /* directory */
S_IFCHR     0020000    /* character special */
S_IFBLK     0060000    /* block special */
S_IFREG     0100000    /* regular */
S_IFIFO     0010000    /* fifo */
S_IFNAM     0050000    /* name special entry */
S_INSEM     01         /* semaphore */
S_INSHD     02         /* shared memory */
S_ISUID     04000      /* set user id on execution */
S_ISGID     02000      /* set group id on execution */
S_ISVTX     01000      /* save swapped text even after use */
S_IREAD     00400      /* read permission, owner */
S_IWRITE    00200      /* write permission, owner */
S_IEXEC     00100      /* execute/search permission, owner */
```

Files

/usr/include/sys/stat.h

See Also

stat(S)

Standards Conformance

stat is conformant with:

AT&T SVID Issue 2, Select Code 307-127;
The X/Open Portability Guide II of January 1987;
IEEE POSIX Std 1003.1-1988 with C Standard Language-Dependent System Support;
and NIST FIPS 151-1.

stune

local tunable parameter file

Syntax

/etc/conf/cf.d/stune

Description

The *stune* file contains local system settings for tunable parameters. The parameter settings in this file replace the default values specified in the *mtune* file, if the new values are within the legal range for the parameter specified in *mtune*. The file contains one line for each parameter to be reset. Each line contains two whitespace-separated fields:

1. *external name*: This is the external name of the tunable parameter used in the *mtune* file.

2. *value*: This field contains the new value for the tunable parameter.

The file *stune* normally resides in */etc/conf/cf.d*. However, a user or an add-on package should never directly edit the *mtune* file. Instead the *idtune* command should be used.

In order for the new values to become effective the UNIX kernel must be rebuilt and the system must then be rebooted.

See Also

mtune(F), idbuild(ADM), idtune(ADM)

syms

common object file symbol table format

Syntax

#include <syms.h>

Description

Common object files contain information to support symbolic software testing [see *sdb*(CP)]. Line number entries, *linenum*(F), and extensive symbolic information permit testing at the C *source* level. Every object file's symbol table is organized as shown below.

```
File name 1.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 1.

File name 2.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 2.
...

Defined global symbols.
Undefined global symbols.
```

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure is given below.

```
#define  SYMNMLEN  8
#define  FILNMLEN  14
#define  DIMNUM    4

struct  syment
{
    union                           /* all ways to get symbol name */
    {
        char       _n_name[SYMNMLEN]; /* symbol name */
        struct
        {
            long    _n_zeroes;   /* == 0L when in string table */
            long    _n_offset;   /* location of name in table */
        } _n_n;
        char        *_n_nptr[2]; /* allows overlaying */
    } _n;
    long           n_value;      /* value of symbol */
    short          n_scnum;      /* section number */
    unsigned short n_type;       /* type and derived type */
    char           n_sclass;     /* storage class */
    char           n_numaux;     /* number of aux entries */
};

#define  n_name    _n._n_name
#define  n_zeroes  _n._n_n._n_zeroes
#define  n_offset  _n._n_n._n_offset
#define  n_nptr    _n._n_nptr[1]
```

Meaningful values and explanations for them are given in both **syms.h** and *Common Object File Format*. Anyone who needs to interpret the entries should seek more information in these sources. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent
{
        struct
        {
                long            x_tagndx;
                union
                {
                        struct
                        {
                                unsigned short  x_lnno;
                                unsigned short  x_size;
                        } x_lnsz;
                        long    x_fsize;
                } x_misc;
                union
                {
                        struct
                        {
                                long   x_lnnoptr;
                                long   x_endndx;
                        }       x_fcn;
                        struct
                        {
                                unsigned short  x_dimen[DIMNUM];
                        }       x_ary;
                }               x_fcnary;
                unsigned short  x_tvndx;
        }       x_sym;
        struct
        {
                char    x_fname[FILNMLEN];
        }       x_file;
         struct
         {
                 long    x_scnlen;
                 unsigned short  x_nreloc;
                 unsigned short  x_nlinno;
         }       x_scn;

        struct
        {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        }       x_tv;
};
```

Indexes of symbol table entries begin at *zero*.

See Also

sdb(CP), a.out(F), linenum(F).

Notes

On machines on which **ints** are equivalent to **longs**, all **longs** have their type changed to **int.** Thus the information about which symbols are declared as **longs** and which, as **ints**, does not show up in the symbol table.

sysfiles

format of UUCP Sysfiles file

Description

The **/usr/lib/uucp/Sysfiles** file lets you assign different files to be used by *uucp*(C) and *cu*(C) as **Systems, Devices**, and **Dialers** files.

You can use different **Systems** files so that requests for login services can be made to different addresses than UUCP services.

With different **Dialers** files you can use different handshaking for *cu* and *uucp*. Multiple **Systems, Dialers**, and **Devices** files are useful if any one file becomes too large.

An active **Sysfiles** file is not included in the distribution. Instead a **Sysfiles.eg** file is included, which contains comments and commented examples of how such a file can be used. This is done because UUCP runs faster without reading this file.

The format of the **Sysfiles** file is

service=w systems=x:x dialers=y:y devices=z:z

where *w* is replaced by *uucico*(ADM), *cu*, or both separated by a colon; *x* is one or more files to be used as the **Systems** file, with each file name separated by a colon and read in the order presented; *y* is one or more files to be used as the **Dialers** file; and *z* is one or more files to be used as the **Devices** file. Each file is assumed to be relative to the **/usr/lib/uucp** directory, unless a full path is given. A backslash-carriage return (\<*CR*>) can be used to continue an entry on to the next line.

An example of using a local *Systems* file in addition to the usual *Systems* file follows:

```
service=uucico:cu  systems=Systems:Local_Systems
```

If this is in */usr/lib/uucp/Sysfiles*, then both **uucico** and **cu** will first look in */usr/lib/uucp/Systems*. If the system they're trying to call doesn't have an entry in that file, or if the entries in the file fail, then they'll look in */usr/lib/uucp/Local_Systems*.

When different *Systems* files are defined for **uucico** and **cu** services, your machine will store two different lists of Systems. You can print the **uucico** list using the *uuname* command or the *cu* list using the *uuname -c* command.

Examples

The following example uses different **Systems** and **Dialers** files to separate the *uucico* and *cu*-specific info, with information that they use in common still in the ''usual'' **Systems** and **Dialers** files.

```
service=uucico   systems=Systems.cico:Systems \
                 dialers=Dialers.cico:Dialers
service=cu       systems=Systems.cu:Systems \
                 dialers=Dialers.cu:Dialers
```

This next example uses the same systems files for uucico and cu, but has split the **Systems** file into local, company-wide, and global files.

```
service=uucico   systems=Systems.local:Systems.company:Systems
service=cu       systems=Systems.local:Systems.company:Systems
```

See Also

uucico(ADM), uucp(C), systems(F)

systemid

the Micnet system identification file

Description

The **systemid** file contains the machine and site names for a system in a Micnet network. A *machine name* identifies a system and distinguishes it from other systems in the same network. A *site name* identifies the network to which a system belongs and distinguishes the network from other networks in the same chain.

The **systemid** file may contain a *site name* and up to four different *machine names*. The file has the form:

```
[site-name]
[machine-name1]
[machine-name2]
[machine-name3]
[machine-name4]
```

The file must contain at least one machine name. The other machine names are optional, serving as alternate names for the same machine. The file must contain a site name if more than one machine name is given or if the network is connected to another through a uucp link. The site name, when given, must be on the first line.

Each name can have up to eight letters and numbers but must always begin with a letter. There is never more than one name to a line. A line beginning with a pound sign (#) is considered a comment line and is ignored.

The Micnet network requires one **systemid** file on each system in a network with each file containing a unique set of machine names. If the network is connected to another network through a uucp link, each file in the network must contain the same site name.

The **systemid** file is used primarily during resolution of aliases. When aliases contain site and/or machine names, the name is compared with the names in the file and removed if there is a match. If there is no match, the alias (and associated message, file, or command) is passed on to the specified site or machine for further processing.

Files

/etc/systemid

See Also

aliases(M), netutil(ADM), top(F)

Value Added

systemid is an extension of AT&T System V provided by the Santa Cruz Operation.

systems

format of UUCP Systems file

Description

The **Systems** file (**/usr/lib/uucp/Systems**) contains the information needed by the *uucico* daemon to establish a communication link to a remote computer. Each entry in the file represents a computer that your computer can call. You can configure the **Systems** file to prevent unauthorized computers from logging in on your computer. More than one entry may be present for a particular computer. These additional entries represent alternative communication paths which the computer tries in sequential order.

Each entry in the *Systems* file has the following format:

sitename schedule device speed phone login-script

sitename field contains the node name of the remote computer.

schedule field is a string that indicates the day-of-week and time-of-day when the remote computer can be called.

device is the device type that should be used to establish the communication link to the remote computer.

speed indicates the transfer speed of the device used in establishing the communication link.

phone provides the phone number of the remote computer for automatic dialers. If you wish to create a portable *Systems* file that can be used at a number of sites where the dialing prefixes differ, see the *dialcodes*(F) man page.

login-script contains login information (also known as a ‘‘chat script’’).

See Also

uucico(ADM), uucp(C), devices(F), dialers(F)

tables

MMDF name tables for aliases, domains, and hosts

Description

All of the MMDF name tables are encoded into a database which is built on top of the *dbm*(S) package. A number of tables are associated with MMDF, the exact set being specified by the tailor file, */usr/mmdf/mmdftailor*. Name tables all have the same format. Functionally, they permit a simple key/value pairing. The syntax for tables is specified here:

entries ::=	entries entry
entry ::=	comment / real-entry
comment ::=	'#' value eol
real-entry ::=	name separator value eol
name ::=	{string of chars not containing a <separator>}
separator ::=	{see the chars in _hkeyend[], usually ':' and space}
value ::=	{string of chars not containing an <eol>}
eol ::=	{see the chars in _hvalend[]}
where:	
name is	a key
value is	any relevant text.

Hosts and Domains

Two basic types of table are host and domain tables. This section gives a brief discussion of these concepts in terms of the MMDF system. The domain namespace is treated as a logical global hierarchy, according to the model of RFC 819, with subdomains separated by '.'s (e.g ISI.USC.ARPA is a three level hierarchy with ARPA at the top level). A host is a computer associated with a channel which may be directly connected or reached through a relay associated with the channel. The distinction between hosts as physical entities, and domains as logical entities should be noted. All hosts known to an MMDF system must have unique names. For this reason, the convention of labelling hosts by an associated domain name is adopted in many cases. This is a useful method to guarantee unique names, but is not required. The domain and host table structures are devised with three basic aims in mind:

1. To map a string into a fully expanded domain name.

2. To map this domain into a source route starting with a host.

3. To obtain the transport address associated with the host.

Domain Tables

Domains are split in a two-level manner, with the top part of the tree specified in the tailor file and the lower parts of the tree in tables. The two level structure is intended as a balance between generality and efficiency. The order of searching is also specified in the tailor file. The structure of a domain table is to have *name* as the part of the domain not in the tailor file. Thus for ISI.USC.ARPA there might be a domain ARPA with name=isi.usc or domain USC.ARPA with name=isi. The structure of value is:

```
value ::=          *(domain dm_separator) host
```

The possible values of dm_separator are given in tai(S), although typically ',' or ' ' would be used. This value is essentially a source route to be traversed from right to left. Consider an example table for the domain ARPA:

```
# Sample ARPA domain table
isi.usc:a.isi.usc.arpa
b.isi.usc:b.isi.usc.arpa
foobar.isi.usc:b.isi.usc.arpa
graphics.isi.usc:graphics.isi.usc.arpa z.mit.arpa
```

Thus, if the "isi.usc.arpa" is analyzed, domain table ARPA will be selected, and host "a.isi.usc.arpa" associated with domain "isi.usc.arpa." If only "isi.usc" were given, the domain tables would be searched in order, and if the ARPA table were the first one to give a match, then the same result would be reached. If "foobar.isi.usc" is given, it would be mapped to host "b.isi.usc.arpa" and (official) domain "b.isi.usc.arpa." If "graphics.isi.usc.arpa" is given, a source route to domain "graphics.isi.usc.arpa" through HOST "z.mit.arpa" will be identified. If "xy.isi.usc.arpa" (or "xy.isi.usc") is given, then it will not be found. However, a subdomain will be stripped from the left and the search repeated. Thus domain "xy.isi.usc.arpa" will be identified as reached by a source route through host "a.isi.usc.arpa."

As specified earlier, the order of searching is also specified in the tailor file. For example, a host in domain UCL-CS.AC.UK, might have a search order UCL-CS.AC.UK, AC.UK, UK, SWEDEN, ARPA, "". Thus, if there were a domain isi.usc.ac.uk, it would be the preferred mapping for isi.usc over isi.usc.arpa. The last domain searched is null. This could be used to contain random fully qualified domains or to identify gateways to other domains. An example file is:

```
#  Sample Top level domain table
#  Odd host
basservax.australia:basservax.australia scunthorpe.ac.uk
# UUCP Gateway
uucp:seismo.arpa
# Mailnet Gateway  (-> multics -> educom ->mailnet)
mailnet:educom.mailnet mit-multics.arpa
```

To specify the top domain in the tailor file, the name and dmn parameters of the domain should be set to "".

Host Tables

For every host associated with the channel, there will be one or more entries. In each case, the key is the name identified from the domain tables. A host may have multiple entries if it has more than one transport address which the channel might utilise.

When a channel just sends all its mail to a relaying site, the address portion (value) of the entry is not needed, directly, during the transmission process. Hence, it need not be accurate. However, it still is used to logically collect together host names, that is, all table entries with the same value are regarded as being aliases for the same host.

P.O. Box Channels

POBox channels, for passive, telephone-based exchange, operate in two modes. In one case, a single login is authorized to pickup all mail for the channel. In this case, the host-table addresses are only used for the "collecting" function. For the second mode, different logins share the channel and are to receive only some of the mail queued for the channel. In this case, the login is treated as an "address", and the table entries should have the value fields contain the name of the login authorized to pickup mail for that "host".

Access control tables

Channels also have access control tables associated with them, to determine whether a message is allowed to use a given route. Each channel has four (optional) tables that determine the access controls used: insrc, outsrc, indest, and outdest.

Reformatting tables

There may also be a ‘‘known hosts’’ table associated with each channel. This is exactly the same format as a host table. If a message is being reformatted, and if an address does not have its host in this list, then it will be modified to appear as a percent route (RFC733 or JNT Mail route) address, with the local domain as the root.

Local Aliases

The password file specifies the name of all local recipients; their mailing names are their login names. Since this is a rather restricted name space, and since it is useful to have some other kinds of locally-known names, there is a second file used to specify "aliases". The location of the aliases file is specified in the tailor file.

An alias entry may be used for one of five functions:

1. True aliasing, where the key value maps to a local user’s login name, e.g. ‘‘dave:dcrocker;’’

2. Forwarding, where the key value maps to a foreign address, such as ‘‘dcrocker:dcrocker@udel;’’ and

3. Address lists, where the key value maps to a set of addresses, such as ‘‘mother:cotton,dcrocker,farber.’’

4. Redirection of a message to a file. For example, ‘‘foobar:dpk/foobar’’ would cause user and group ids to be set to dpk and the text of the message to be appended to the file ‘‘foobar’’ in dpk’s default login directory. Similarly, ‘‘foobar:dpk//tmp/foobar’’ would do the same for file /tmp/foobar.

5. Redirection of a message to a pipe. For example, ‘‘news-inject:newsl/usr/lib/news/uurec’’ would cause a message to be passed into a UNIX pipe (see *pipe*(S)) with userid and groupid set to news.

As a convenience, the value-part of an entry may specify a file name, so that the *actual* value is taken from the file. There are two possible notations for this:

1. By having left-angle bracket ('<') precede the value specification. For example: ``mother: < /etc/mmdf/mother_list@udel-relay.arpa.''

2. By using a data type with value ``include.'' For example: ``mother: :include: /etc/mmdf/mother@udel-relay.arpa''

In both cases, the @HOST (not a domain) is optional. If specified, it should be the local host.

Recursive specification is permitted. For example, ``crocker'' may map to ``dcrocker'' and ``dcrocker'' may map to ``dcrocker at udel,'' so that both ``crocker'' and ``dcrocker'' are locally-known names, but mail sent to either of them will be forwarded to ``dcrocker@udel.''

In practice, it is useful to organize alias files into the following ordering:

List aliases
: which contain a value referring to a later address list. This constitutes a one-to-one mapping of a key to a value, where the value points into the ``Lists'' group.

Lists
: which contain values referring to multiple addresses; This constitutes a one-to-many mapping, where some of the addresses may refer to other entries (address lists) in the Lists group, as well as other entries (individual addresses) later in the table.

Mailbox aliases
: which contain values referring to single addresses. These constitute one-to-one mappings, where the value refers to an entry in the password file or to an entry in the ``Forwarding aliases'' group.

Forwarding aliases
: which contain values referring to single addresses on other machines. These, also, are one-to-one mappings, where the value always refers to an off-machine address.

By organizing the file in this manner, only the ``Lists'' portion requires a topological sort. Since the other three sections will never point to entries within their section, they may be sorted more conveniently, such as alphabetically. Such a structure also tends to make changes easy. In particular, the handling of forwarding is easy, since *all* references to a user will get intercepted, at the end of the table.

Mail-ID tables

The Mail-ID tables are used only if the Mail-IDs feature is enabled. This can be done in the tailoring file, by defining MMAILID to be 1. Mail-IDs are used to disassociate mail addresses from login names.

There are two tables that are used to map Mail-IDs to users login names and login ids to Mail-IDs. The ''users'' file is used to map users (login ids) to Mail-IDs, and the ''mailids'' file is used to map Mail-IDs to users. The names of these files can be overridden, but it is not recommended. Each file has lines with two entries per line (user and Mail-ID, or Mail-ID and user).

A user can have more than one entry in the Mail-IDs file, but should have only one entry in the users file. This does not prevent them from sending mail with any of their Mail-IDs. The users file is just a source of default Mail-IDs.

Value Added

tables is an extension of AT&T System V provided by the Santa Cruz Operation.

tar

archive format

Description

The command *tar*(C) dumps files to and extracts files from backup media or the hard disk.

Each file is archived in contiguous blocks, the first block being occupied by a header, whose format is given below, and the subsequent blocks of the files occupying the following blocks. All headers and file data start on 512 byte block boundaries and any spare unused space is padded with garbage. The format of a header block is as follows:

```
#define TBLOCK 512
#define NBLOCK 20
#define NAMSIZ 100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
                char extno[4];
                char extotal[4];
                char efsize[12];
        } dbuf;
} dblock;
```

The name entry is the path name of the file when archived. If the pathname starts with a zero word, the entry is empty. It is at most 100 bytes long and ends in a null byte. Mode, uid, gid, size, and time modified are the same as described under i-nodes (refer to *filesystem* (F)). The checksum entry has a value such that the sum of the words of the directory entry is zero.

If the entry corresponds to a link, then *linkname* contains the pathname of the file to which this entry is linked and *linkflag* is set to 0 if there are no links, or 1 if there are links. No data is put in the archive file.

See Also

filesystem(F), tar(C)

Standards Conformance

tar is conformant with:
AT&T SVID Issue 2, Select Code 307-127.

term

terminal driving tables for nroff

Description

nroff(CT) uses driving tables to customize its output for various types of output devices, such as printing terminals, special word-processing printers (such as Diablo, Qume, or NEC Spinwriter mechanisms), or special output filter programs. These driving tables are written as C programs, compiled, and installed in **/usr/lib/term/tab***name*, where *name* is the name for that terminal type as shown in **term**(CT).

The structure of the tables is as follows. Sizes are in 240ths of an inch.

```
#define   INCH      240
#include /usr/lib/term/terms.h

struct termtable tlp ; { \* lp is the name of the term, *\
          int bset; \* modify with new name, such as tnew *\
          int breset;
          int Hor;
          int Vert;
          int Newline;
          int Char;
          int Em;
          int Halfline;
          int Adj;
          char *twinit;
          char *twrest;
          char *twnl;
          char *hlr;
          char *hlf;
          char *flr;
          char *bdon;
          char *bdoff;
          char *iton;
          char *itoff;
          char *ploton;
          char *plotoff;
          char *up;
          char *down;
          char *right;
          char *left;
          char *codetab[256-32];
          char *zzz;
} ;
```

The meanings of the various fields are as follows:

bset bits to set in *termio.c_oflag* see **tty**(M) and **termio**(M)). after output.

breset bits to reset in *termio.c_oflag* before output.

Hor horizontal resolution in fractions of an inch.

Vert vertical resolution in fractions of an inch.

Newline space moved by a newline (linefeed) character in fractions of an inch.

Char quantum of character sizes, in fractions of an inch. (i.e., characters are multiples of Char units wide. See *codetab* below.)

Em size of an em in fractions of an inch.

Halfline space moved by a half-linefeed (or half-reverse-linefeed) character in fractions of an inch.

Adj quantum of white space for margin adjustment in the absence of the **-e** option, in fractions of an inch. (i.e., white spaces are a multiple of Adj units wide)

Note: if this is less than the size of the space character (in units of Char; see below for how the sizes of characters are defined), *nroff* will output fractional spaces using plot mode. Also, if the **-e** switch to *nroff* is used, Adj is set equal to Hor by *nroff*.

twinit set of characters used to initialize the terminal in a mode suitable for *nroff*.

twrest set of characters used to restore the terminal to normal mode.

twnl set of characters used to move down one line.

hlr set of characters used to move up one-half line.

hlf set of characters used to move down one-half line.

flr set of characters used to move up one line.

bdon set of characters used to turn on hardware boldface mode, if any. *Nroff* assumes that boldface mode is reset automatically by the *twnl* string, because many letter-quality printers reset the boldface mode when they receive a carriage return; the *twnl* string should include whatever characters are necessary to reset the boldface mode.

bdoff set of characters used to turn off hardware boldface mode, if any.

iton set of characters used to turn on hardware italics mode, if any.

itoff set of characters used to turn off hardware italics mode, if any.

ploton set of characters used to turn on hardware plot mode (for Diablo-type mechanisms), if any.

plotoff set of characters used to turn off hardware plot mode (for Diablo-type mechanisms), if any.

up set of characters used to move up one resolution unit (Vert) in plot mode, if any.

down set of characters used to move down one resolution unit (Vert) in plot mode, if any.

right set of characters used to move right one resolution unit (Hor) in plot mode, if any.

left set of characters used to move left one resolution unit (Hor) in plot mode, if any.

codetab Array of sequences to print individual characters. Order is *nroff*'s internal ordering. See the file **/usr/lib/term/tabuser.c** for the exact order.

zzz a zero terminator at the end.

The *codetab* sequences each begin with a flag byte. The top bit indicates whether the sequence should be underlined in the **.ul** font. The rest of the byte is the width of the sequence in units of *Char*.

The remainder of each *codetab* sequence is a sequence of characters to be output. Characters with the top bit off are output as given; characters with the top bit on indicate escape into plot mode. When such an escape character is encountered, *nroff* shifts into plot mode, emitting *ploton*, and skips to the next character if the escape character was '\200'.

When in plot mode, characters with the top bit off are output as given. A character with the top bit on indicates a motion. The next bit indicates coordinate, with **1** being vertical and **0** being horizontal.
The next bit indicates direction, with **1** meaning up or left. The remaining five bits give the amount of the motion. An amount of zero causes exit from plot mode.

When plot mode is exited, either at the end of the string or via the amount-zero exit, *plotoff* is emitted followed by a blank.

All quantities which are in units of fractions of an inch should be expressed as INCH**num/denom*, where *num* and *denom* are respectively the numerator and denominator of the fraction; that is, 1/48 of an inch would be written as ''INCH/48''.

If any sequence of characters does not pertain to the output device, that sequence should be given as a null string.

The Development System must be installed on the computer to create a new driving table. The source code for a generic output device is in the file **/usr/lib/term/tabuser.c** Copy this file and make the necessary modifications, including the name of the termtable struct. Refer to the hardware manual for the codes needed for the output device (terminal, printer, etc.). Name the file according to the convention explained in **term**(CT). The makefile, **/usr/lib/term/makefile**, should be updated to include the source file to the new driving table. To perform the modification, enter the command:

```
cc -M3e -O -c tabuser.c maketerm.o -o maketerm
```

When the files are prepared, enter the command :

```
make
```

(See **make**(CP)). The source to the new driving table is linked with the object file **mkterm.o**, and the new driving table is created and installed in the proper directory.

Files

/usr/lib/term/tab*name* driving tables
/usr/lib/term/tabuser.c generic source for driving tables
/usr/lib/term/makefile makefile for creating driving tables
/usr/lib/term/mkterm.olinkable object file for creating driving tables
/usr/lib/term/terms.h used to create nroff driving tables

See Also

nroff(CT), term(CT)

Notes

The Development System and text processing software must be installed on the computer to create new driving tables.

Not all UNIX facilities support all of these options.

termcap

terminal capability data base

Description

The file **/etc/termcap** is a data base describing terminals. This data base is used by commands such as *vi*(C), Lyrix®, Multiplan™ and sub-routine packages such as *curses*(S). Terminals are described in *termcap* by giving a set of capabilities and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap*.

Entries in *termcap* consist of a number of fields separated by colons ':'. The first entry for each terminal gives the names that are known for the terminal, separated by vertical bars (|). For compatibility with older systems the first name is always 2 characters long. The second name given is the most common abbreviation for the terminal and the name used by *vi (C)* and *ex*(C). The last name given should be a long name fully identifying the terminal. Only the last name can contain blanks for readability.

Capabilities (including XENIX Extensions)

The following is a list of the capabilities that can be defined for a given terminal. In this list, (P) indicates padding can be specified, and (P*) indicates that padding can be based on the number of lines affected. The capability type and padding fields are described in detail in the following section ''Types of Capabilities.''

The codes beginning with uppercase letters (except for CC) indicate XENIX extensions. They are included in addition to the standard entries and are used by one or more application programs. As with the standard entries, not all modes are supported by all applications or terminals. Some of these entries refer to specific terminal output capabilities (such as GS for ''graphics start''). Others describe character sequences sent by keys that appear on a keyboard (such as PU for PageUp key). There are also entries that are used to attribute special meanings to other keys (or combinations of keys) for use in a particular software program. Some of the XENIX extension capabilities have a similar function to standard capabilities. They are used to redefine specific keys (such as using function keys as arrow keys). The extension capabilities are included in the **/etc/termcap** file, as they are required for some utilities. The more commonly used extension capabilities are described in more detail in the section ''XENIX Extensions.''

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not **^H**
bs	bool		Terminal can backspace with **^H**
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
CF	str		Cursor off
ch	str	(P)	Like cm but horizontal motion only, line stays same
CL	str		Sent by CHAR LEFT key
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
CO	str		Cursor on
cr	str	(P*)	Carriage return, (default **^M**)
cs	str	(P)	Change scrolling region (vt100), like **cm**
cv	str	(P)	Like **ch** but vertical only.
CW	str		Sent by CHANGE WINDOW key
da	bool		Display may be retained above
DA	bool		Delete attribute string
db	bool		Display may be retained below
dB	num		Number of millisec of **bs** delay needed
dC	num		Number of millisec of **cr** delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of **ff** delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of **nl** delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give `:ei=:´ if **ic**
EN	str		Sent by END key
eo	bool		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default **^L**)
G1	str		Upper-right (1st quadrant) corner character
G2	str		Upper-left (2nd quadrant) corner character

Name	Type	Pad?	Description
G3	str		Lower-left (3rd quadrant) corner character
G4	str		Lower-right (4th quadrant) corner character
GC	str		Center graphics character (similar to "+")
GD	str		Down-tick character
GE	str		Graphics mode end
GG	num		Number of chars taken by GS and GE
GH	str		Horizontal bar character
GL	str		Left-tick character
GR	str		Right-tick character
GS	str		Graphics mode start
GU	str		Up-tick character
GV	str		Vertical bar character
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
HM	str		Sent by HOME key (if not **kh**)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	str		Insert mode (enter); give ':im=' if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by 'other' function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of 'keypad transmit' mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of 'other' keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in 'keypad transmit' mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on 'other' function keys
LD	str		Sent by line delete key
LF	str		Sent by line feed key
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by **vi** version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor
MP	str		Multiplan initialization string
MR	str		Multiplan reset string
ms	bool		Will scroll in stand-out mode
mu	str		Memory unlock (turn off memory lock)

Name	Type	Pad?	Description
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll
NU	str		Sent by NEXT UNLOCKED CELL key
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
PD	str		Sent by PAGE DOWN key
PN	str		Start local printing
PS	str		End local printing
pt	bool		Has hardware tabs (may need to be set with **is**)
PU	str		Sent by PAGE UP key
RC	str		Sent by RECALC key
RF	str		Sent by TOGGLE REFERENCE key
RT	str		Sent by RETURN key
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by **so** or **se**
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than **^I** or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by **us** or **ue**
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
UP	str		Sent by up-arrow key (alternate to ku)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
WL	str		Sent by WORD LEFT key
WR	str		Sent by WORD RIGHT key
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like **ce \r \n** (Delta Data)
xs	bool		Standard out not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic **so** char (Teleray 1061)

A Sample Entry

The following entry describes the Concept-100, and is among the more complex entries in the *termcap* file. (This particular Concept entry is outdated, and is used as an example only.)

```
c1|c100|concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:\
        :cm=\Ea%+ %+ :co#80:dc=16\E^A:dl=3*\E^B:\
        :ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue over to multiple lines by giving a backslash (\) as the last character of a line. Empty fields can be included for readability between the last field on a line and the first field on the next. Capabilities in *termcap* are of three types: Boolean capabilities, which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence that can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has 'automatic margins' (i.e., an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. The description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co**, which indicates the number of columns the terminal has, gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the rest of the string is sent to provide this delay. The delay can be either a integer, e.g., '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' to specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a colon (**:**) in a capability, it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability, it must be encoded as **\200**. The routines that deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it. To test a new terminal description, you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in **/etc/termcap.** TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor.

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, it should have the **am** capability. If the terminal can clear its screen, this is given by the **cl** string capability. If the terminal can backspace, it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H** in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over), it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on (i.e., **am**).

These capabilities suffice to describe hardcopy and ‘‘glass-tty’’ terminals. Thus the model 33 teletype is described as

```
t3|33|tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as:

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability. This capability uses *printf*(S) like escapes (such as %**x**) in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, its arguments are the line and then the column to which motion is desired, and the % encodings have the following meanings:

%d	replaced by line/column position, 0 origin
%2	like %2d - 2 digit field
%3	like %3d - 3 digit field
%.	like *printf*(S) %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output
%r	reverses order of line and column, no output
%i	increments line/column position (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output
%D	Reverse coding (x-2*(x%16)), no output (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is 'cm=6\E&%r%2c%2Y'. The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, 'cm=^T%.%.'. Terminals that use '%.' need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus 'cm=\E=%+ %+ '.

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, it should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen), this can be given as **ho**; similarly, a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, the sequence should be given as **ce**. If the terminal can clear from the current position to the end of the display, the sequence should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, the sequence should be given as **al**. Note that this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line on which the cursor rests, the sequence should be given as **dl**. This is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, the sequence can be given as **sb**, but **al** can suffice. If the terminal can retain display memory above, the **da** capability should be given, and if display memory can be retained below, then **db** should be given. These let the editor know that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to the insert/delete character that can be described using *termcap*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and entering text separated by cursor motions. Enter 'abc def', using local cursor motions (not spaces) between the 'abc' and the 'def'. Then position the cursor before the 'abc' and put the terminal in insert mode. If entering characters causes the rest of the line to shift rigidly and characters to fall off the end, your terminal does not distinguish between blanks and untyped positions. If the 'abc' shifts over to the 'def' which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for 'insert null'. No known terminals have an insert mode, not falling into one of these two classes.

The editor can handle both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Specify **im** as the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Specify **ei** as the sequence to leave insert mode (specify this with an empty value if you also gave **im** an empty value). Now specify **ic** as any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not support **ic**, terminals that send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode, these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as reverse video, blinking, or underlining - half bright is not usually an acceptable 'standout' mode unless the terminal is in reverse video mode constantly), the preferred mode is reverse video by itself. It is acceptable, if the code to change into or out of standout mode leaves one, or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do. Although it may confuse some programs slightly, it cannot be helped.

Codes to begin underlining and end underlining can be given as **us**, and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, the sequence can be given as **uc**. (If the underline code does not move the cursor to the right, specify the code followed by a nondestructive space.)

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), the sequence can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, the sequence can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the

Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed), even though it does not overstrike, you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by specifying **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not to transmit, enter these codes as **ks** and **ke**. Otherwise, the keypad is assumed always to transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh**. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, ':ko=cl,ll,sf,sb:', which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete, but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the Mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, the sequence can be given as **ta**.

Terminals that do not allow '~' characters to be displayed (such as Hazeltines), should indicate **hz**. Datamedia terminals that echo carriage-return-linefeed for carriage return, and then ignore a following linefeed, should indicate **nc**. Early Concept terminals, that ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **x***x*.

If the leading character for commands to the terminal (normally the escape character) can be set by the software, specify the command character(s) with the capability **CC**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** is displayed before **if**. This is useful where **if** is **/usr/lib/tabset/std** , but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability, **tc**, can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since *termlib* routines search the entry from left to right, and since the **tc** capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with **xx@** where xx is the capability. For example:

```
hn|2621nl:ks@:ke@:tc=2621:
```

This defines a 2621nl that does not have the **ks** or **ke** capabilities, and does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

XENIX Extensions

Capabilities This table lists the (previously listed) XENIX extensions to the termcap capabilities. It shows which codes generate information input from the keyboard to the program reading the keyboard and which codes generate information output from the program to the screen.

Name	Input/Output	Description
CF	str	Cursor off
CL	str	Sent by CHAR LEFT key
CO	str	Cursor on
CW	str	Sent by CHANGE WINDOW key
DA	bool	Delete attribute string
EN	str	Sent by END key
G1	str	Upper-right (1st quadrant) corner character
G2	str	Upper-left (2nd quadrant) corner character
G3	str	Lower-left (3rd quadrant) corner character
G4	str	Lower-right (4th quadrant) corner character
G5	str	Upper right (1st quadrant) corner character (double)
G6	str	Upper left (2nd quadrant) corner character (double)
G7	str	Lower left (3rd quadrant) corner character (double)
G8	str	Lower right (4th quadrant) corner character (double)
GC	str	Center graphics character (similar to +)
Gc	str	Centre graphics character (double)
GD	str	Down-tick character
Gd	str	Down tick character (double)
GE	str	Graphics mode end
GG	num	Number of chars taken by GS and GE
GH	str	Horizontal bar character
Gh	str	Horizontal bar character (double)
GL	str	Left-tick character
Gl	str	left-tick character (double)
GR	str	Right-tick character
Gr	str	right-tick character (double)
GS	str	Graphics mode start
GU	str	Up-tick character
Gu	str	Up-tick character (double)
GV	str	Vertical bar character
Gv	str	Vertical bar character (double)
HM	str	Sent by HOME key (if not **kh**)
mb	str	blinking on
me	str	blinking off
MP	str	Multiplan initialization string
MR	str	Multiplan reset string
NU	str	Sent by NEXT UNLOCKED CELL key
PD	str	Sent by PAGE DOWN key
PU	str	Sent by PAGE UP key
RC	str	Sent by RECALC key
RF	str	Sent by TOGGLE REFERENCE key
RT	str	Sent by RETURN key
UP	str	Sent by up-arrow key (alternate to ku)
WL	str	Sent by WORD LEFT key
WR	str	Sent by WORD RIGHT key

Cursor motion Some application programs make use of special editing codes. **CR** and **CL** move the cursor one character right and left respectively. **WR** and **WL** move the cursor one word right and left respectively. **CW** changes windows, when they are used in the program.

Some application programs turn off the cursor. This is accomplished using **CF** for cursor off and **CO** to turn it back on.

Graphic mode. If the terminal has graphics capabilities, this mode can be turned on and off with the **GS** and **GE** codes. Some terminals generate graphics characters from all keys when in graphics mode (such as the Visual 50). The other **G** codes specify particular graphics characters accessed by escape sequences. These characters are available on some terminals as alternate graphics character sets (not as a bit-map graphic mode). The vt100 has access to this kind of alternate graphics character set, but not to a bit-map graphic mode.

Files

/etc/termcap	File containing terminal descriptions

See Also

ex(C), curses(S), termcap(S), tset(C), vi(C), more(C), screen(HW)

Credit

This utility was developed at the University of California at Berkeley and is used with permission.

Notes

ex(C) allows only 256 characters for string capabilities, and the routines in *termcap(S)* do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the *vi*(C) program.

Not all programs support all entries. There are entries that are not supported by any program.

XENIX termcap extensions are explained in detail in the software application documentation.

Refer to the *screen*(HW) manual page, for a description of the character sequences used by the monitor device on your specific system.

terminfo

format of compiled terminfo file

Description

Compiled terminfo descriptions are placed under the directory **/usr/lib/terminfo**. In order to avoid a linear search of a huge UNIX system directory, a two-level scheme is used: **/usr/lib/terminfo/ c/name** where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, *act4* can be found in the file **/usr/lib/terminfo/ a/act4**. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8- or more-bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the *tic*(C) program, and read by the routine *setupterm* in *terminfo*(S). The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal 0432); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value -1 is represented by 0377, 0377; other negative values are illegal. The -1 generally means that a capability is missing from this terminal. Note that this format corresponds to the hardware of the VAX and PDP-11. Machines in which this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the '|' character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1, as the flag is present or absent. The capabilities are in the same order as the file **<term.h>**.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short-word boundary.

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is -1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of -1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information $<nn> and parameter information %x are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null-terminated.

Note that it is possible for *setupterm* to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since *setupterm* was recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine *setupterm* must be prepared for both possibilities; this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
    cr=^M, cud1=^J, ind=^J, bel=^G, am, cub1=^H,
    ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
    cols#80, lines#24, cuf1=^X, cuu1=^Z, home=^],
```

```
3000 032 001      \0 025  \0  \b  \0 212  \0   "  \0   m   i   c   r
020    o   t   e   r   m   |   a   c   t   4   |   m   i   c   r   o
040    t   e   r   m       a   c   t       i   v  \0  \0 001  \0  \0
060   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
100   \0  \0   P  \0 377 377 030  \0 377 377 377 377 377 377 377 377
120  377 377 377 377  \0  \0 002  \0 377 377 377 377 004  \0 006  \0
140   \b  \0 377 377 377 377  \n  \0 026  \0 030  \0 377 377 032  \0
160  377 377 377 377 034  \0 377 377 036  \0 377 377 377 377 377 377
200  377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520  377 377 377 377      \0 377 377 377 377 377 377 377 377 377 377
540  377 377 377 377 377 377 007  \0  \r  \0  \f  \0 036  \0 037  \0
560  024   %   p   1   %   c   %   p   2   %   c  \0  \n  \0 035  \0
600   \b  \0 030  \0 032  \0  \n  \0
```

Some limitations: the total size of a compiled description cannot exceed 4096 bytes; the name field cannot exceed 128 bytes.

Files

/usr/lib/terminfo/*/*	compiled terminal capability data base

See Also

terminfo(M), terminfo(S), tic(C)

timezone

set default system time zone

Syntax

/etc/TIMEZONE

Description

This file sets and exports the time zone environmental variable **TZ**.

This file is ‘‘dotted’’ into other files that must know the time zone, including **/etc/cshrc**, **/etc/profile**, **/etc/rc2**, **.profile**.

TZ contains the following information:

(*sss*) One to nine letters designating the standard time zone.

(*n*) Number of hours past Greenwich mean time for the standard time (partial hours are valid e.g. 12:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich.

(*ddd*) One to nine letters designating the local daylight savings time (summer time) zone. If not present, summer time is assumed not to apply.

(*m*) Number of hours past Greenwich mean time for the summer time (partial hours are valid e.g. 11:30:01). Positive hours are west of Greenwich, negative numbers are east of Greeenwich. If *m* is not given, the distance to GMT during summer time is assumed to be one hour less than during standard time.

(*start*) The rule defining the day summer time begins. In the southern hemisphere, the ending day will be earlier in the year than the starting day.

(*end*) The rule defining the day summer time ends.

(*time*) The time of day the change to and from summer time occurs. The default is 02:00:00 local time.

The rules for defining the **start** and **end** of summer time are as follows:

J*n*	1 based Julian day n ($1 \le n \le 365$)*
n	0 based Julian day n ($0 \le n \le 364$)*
W*n.d*	day d ($0 \le d \le 6$)** of week n ($1 \le n \le 53$)†
M*m.n.d*	day d of week n ($1 \le n \le 5$)‡ of month m ($1 \le m \le 12$)

* Leap days (February 29) are never counted; that is, February 28 (J59) is immediately followed by March 1 (J60) even in leap years.

** Sunday is the first day of the week (0). If *d* is omitted, Sunday is assumed. Note that *d* is optional.

† The 5th week of the month is always the last week containing day *d*, whether there are actually 4 or 5 weeks containing day *d*.

‡ The 53rd week of the year is always the last week containing day *d*, whether there are actually 52 or 53 weeks containing day *d*.

If **start** and **end** are omitted, current U.S. law is assumed.

Examples

A simple setting for New Jersey could be

```
TZ=’EST5EDT’
```

where **EST** is the abbreviation for the main time zone, **5** is the difference, in hours, between GMT (Greenwich Mean Time) and the main time zone, and **EDT** is the abbreviation for the alternate time zone.

The most complex representation of the same setting, for the year 1986, is

```
TZ=’EST5:00:00EDT4:00:00;117/2:00:00,299/2:00:00’
```

where **EST** is the abbreviation for the main time zone, **5:00:00** is the difference, in hours, minutes, and seconds between GMT and the main time zone, **EDT** is the abbreviation for the alternate time zone, **4:00:00** is the difference, in hours, minutes, and seconds between GMT and the alternate time zone, **117** is the number of the day of the year (Julian day) when the alternate time zone will take effect, **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will take effect, **299** is the number of the day of the year when the alternate time zone will end, and **2:00:00** is the number of hours, minutes, and seconds past midnight when the alternate time zone will end.

A southern hemisphere setting such as the Cook Islands could be

```
TZ='KDT9:30KST10:00;64/5:00,303/20:00'
```

This setting means that **KDT** is the abbreviation for the main time zone, **KST** is the abbreviation for the alternate time zone, KST is **9** hours and **30** minutes later than GMT, KDT is **10** hours later than GMT, the starting date of KDT is the **64**th day at **5** AM, and the ending date of KDT is the **303**rd day at **8** PM.

Starting and ending times are relative to the alternate time zone. If the alternate time zone start and end dates and the time are not provided, the days for the United States that year will be used and the time will be 2 AM. If the start and end dates are provided but the time is not provided, the time will be midnight.

Note that in most installations, **TZ** is set to the correct value by default when the user logs on, via the local */etc/profile* file [see *profile*(F)].

See Also

ctime(S), profile(F), environ(M), TZ(M), rc2(ADM)

Notes

Setting the time during the interval of change from the main time zone to the alternate time zone or vice versa can produce unpredictable results.

Standards Conformance

timezone is conformant with:

The X/Open Portability Guide II of January 1987.

top, top.next

the Micnet topology files

Description

These files contain the topology information for a Micnet network. The topology information describes how the individual systems in the network are connected, and what path a message must take from one system to reach another. Each file contains one or more lines of text. Each line of text defines a connection or a communication path.

The **top** file defines connections between systems. Each line lists the machine names of the connected systems, the serial lines used to make the connection, and the speed (baud rate) of transmission between the systems. Each line has the following format:

```
machine1 tty1a machine2 tty2a speed
```

machine1 and *machine2a* are the machine names of the respective systems (as given in the **systemid** files). The *ttys* are the device names (e.g., tty1a) of the connecting serial lines. The speed must be an acceptable baud rate (e.g., 110, 300, ..., 19200).

The **top.next** file contains information about how to reach a particular system from a given system. There may be several lines for each system in the network. Each line lists the machine name of a system, followed by the machine name of a system connected to it, followed by the machine names of all the systems that may be reached by going through the second system. Such a line has the form:

```
machine1 machine2 machine3 [machine4]...
```

The machine names must be the names of the respective systems (as given by the first machine name in the **systemid** files).

The *top.next* file must be present even if there are only two computers in the network. In such a case, the file must be empty.

In the **top** and **top.next** files, any line beginning with a number sign (#) is considered a comment, and is ignored.

Files

/usr/lib/mail/top

/usr/lib/mail/top.next

See Also

netutil(ADM), systemid(F)

types

primitive system data types

Syntax

#include <sys/types.h>

Description

The data types defined in the include file **<sys/types.h>** are used in UNIX system code; some data of these types are accessible to user code.

The form *daddr_t* is used for disk addresses except in an inode on disk, see *filesystem* (F). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

See Also

filesystem(F)

Standards Conformance

types is conformant with:

The X/Open Portability Guide II of January 1987.

unistd

file header for symbolic constants

Syntax

#include <unistd.h>

Description

The header file <*unistd.h*> lists the symbolic constants and structures not already defined or declared in some other header file.

```
/* Symbolic constants for the "access" routine: */

#define R_OK       4     /*Test for Read permission */
#define W_OK       2     /*Test for Write permission */
#define X_OK       1     /*Test for eXecute permission */
#define F_OK       0     /*Test for existence of File */

#define F_ULOCK    0     /*Unlock a previously locked region */
#define F_LOCK     1     /*Lock a region for exclusive use */
#define F_TLOCK    2     /*Test and lock a region for exclusive use */
#define F_TEST     3     /*Test a region for other processes locks */

/*Symbolic constants for the "lseek" routine: */

#define SEEK_SET   0     /* Set file pointer to "offset" */
#define SEEK_CUR   1     /* Set file pointer to current plus "offset" */
#define SEEK_END   2     /* Set file pointer to EOF plus "offset" */

/*Path names:*/

#define GF_PATH    "/etc/group"    /*Path name of the group file */
#define PF_PATH    "/etc/passwd"   /*Path name of the passwd file */
```

Standards Conformance

unistd is conformant with:

The X/Open Portability Guide II of January 1987.

utmp, wtmp

formats of utmp and wtmp entries

Syntax

#include <sys/types.h>
#include <utmp.h>

Description

These files, which hold user and accounting information for such commands as *who*(C), *write*(C), and *login*(M), have the following structure as defined by **<utmp.h>**:

```
#define   UTMP_FILE   ''/etc/utmp''
#define   WTMP_FILE   ''/etc/wtmp''
#define   ut_name     ut_user

struct utmp {
       char       ut_user[8];         /* User login name */
       char       ut_id[4];           /* usually line # */
       char       ut_line[12];        /* device name (console, lnxx) */
       short      ut_pid;             /* process id */
       short      ut_type;            /* type of entry */
       struct     exit_status {
           short      e_termination;  /* Process termination status */
           short      e_exit;         /* Process exit status */
       } ut_exit;                     /* The exit status of a process
                                         marked as DEAD_PROCESS. */
       time_t     ut_time;            /* time entry was made */
};

/*  Definitions for ut_type  */

#define EMPTY                 0
#define RUN_LVL                   1
#define BOOT_TIME             2
#define OLD_TIME              3
#define NEW_TIME              4
#define INIT_PROCESS          5   /* Process spawned by "init" */
#define LOGIN_PROCESS      6  /* A "getty" process waiting for login */
#define USER_PROCESS          7   /* A user process */
#define DEAD_PROCESS          8
#define ACCOUNTING            9
#define UTMAXTYPE ACCOUNTING /* Largest legal value of ut_type */
```

```
/*  Special strings or formats used in the "ut_line" field when  */
/*  accounting for something other than a process  */
/*  No string for the ut_line field can be more than 11 chars +  */
/*  a NULL in length  */
#define RUNLVL_MSG "run-level %c"
#define BOOT_MSG   "system boot"
#define OTIME_MSG  "old time"
#define NTIME_MSG  "new time"
```

Files

/usr/include/utmp.h
/etc/utmp
/etc/wtmp

See Also

getut(S), login(M), who(C), write(C)

Standards Conformance

utmp and *wtmp* are conformant with:

The X/Open Portability Guide II of January 1987.

x.out

format of XENIX link editor output

Syntax

#include <x.out.h>

Description

The output of the XENIX link editor, called the *x.out* or segmented *x.out* format, is defined by the files **/usr/include/x.out.h** and **/usr/include/sys/relsym.h**. The *x.out* file has the following general layout:

1. Header.
2. Extended header.
3. File segment table (for segmented formats).
4. Segments (Text, Data, Symbol, and Relocation).

In the segmented format, there may be several text and data segments, depending on the memory model of the program. Segments within the file begin on boundaries which are multiplies of 512 bytes as defined by the file's pagesize.

Format

```
/*
 * The main and extended header structures.
 * For x.out segmented (XE_SEG):
 *     1) fields marked with (s) must contain sums of xs_psize for
 *    non-memory images, or xs_vsize for memory images.
 *     2) the contents of fields marked with (u) are undefined.
 */

struct xexec {              /* x.out header */
   unsigned short  x_magic; /* magic number */
   unsigned short  x_ext;    /* size of header extension */
   long      x_text;      /* size of text segment (s) */
   long      x_data;      /* size of initialized data (s) */
   long      x_bss;   /* size of uninitialized data (s) */
   long      x_syms;      /* size of symbol table (s) */
   long      x_reloc; /* relocation table length (s) */
   long      x_entry; /* entry point, machine dependent */
```

```
    char      x_cpu;    /* cpu type & byte/word order */
    char      x_relsym; /* relocation & symbol format (u) */
    unsigned short  x_renv;      /* run-time environment */
};


struct xext {                  /* x.out header extension */
    long      xe_trsize;       /* size of text relocation (s) */
    long      xe_drsize;       /* size of data relocation (s) */
    long      xe_tbase;    /* text relocation base (u) */
    long      xe_dbase;    /* data relocation base (u) */
    long      xe_stksize;      /* stack size (if XE_FS set) */
              /* the following must be present if XE_SEG */
    long      xe_segpos;       /* segment table position */
    long      xe_segsize;      /* segment table size */
    long      xe_mdtpos;   /* machine dependent table position */
    long      xe_mdtsize;  /* machine dependent table size */
    char      xe_mdttype;  /* machine dependent table type */
    char      xe_pagesize; /* file pagesize, in multiples of 512 */
    char      xe_ostype;       /* operating system type */
    char      xe_osvers;       /* operating system version */
    unsigned short  xe_eseg;  /* entry segment, machine dependent */
    unsigned short  xe_sres;  /* reserved */
};


struct xseg {                 /* x.out segment table entry */
    unsigned short  xs_type;  /* segment type */
    unsigned short  xs_attr;  /* segment attributes */
    unsigned short  xs_seg;       /* segment number */
    char      xs_align;    /* log base 2 of alignment */
    char      xs_cres;     /* unused */
    long      xs_filpos;       /* file position */
    long      xs_psize;    /* physical size (in file) */
    long      xs_vsize;    /* virtual size (in core) */
    long      xs_rbase;    /* relocation base address/offset */
    unsigned short  xs_noff;  /* segment name string table offset */
    unsigned short  xs_sres;  /* unused */
    long      xs_lres;     /* unused */
};


struct xiter {                /* x.out iteration record */
    long      xi_size;     /* source byte count */
    long      xi_rep;      /* replication count */
    long      xi_offset;   /* destination offset in segment */
};

struct xlist {                /* xlist structure for xlist(3). */
    unsigned short  xl_type;  /* symbol type */
    unsigned short  xl_seg;       /* file segment table index */
    long      xl_value;        /* symbol value */
    char      *xl_name;        /* pointer to asciz name */
};

struct aexec {                /* a.out header */
```

```
        unsigned short  xa_magic;     /* magic number */
        unsigned short  xa_text;      /* size of text segment */
        unsigned short  xa_data;      /* size of initialized data */
        unsigned short  xa_bss;       /* size of uninitialized data */
        unsigned short  xa_syms;      /* size of symbol table */
        unsigned short  xa_entry;     /* entry point */
        unsigned short  xa_unused;    /* not used */
        unsigned short  xa_flag;      /* relocation info stripped */
};


struct nlist {                  /* nlist structure for nlist(3). */
        char     n_name[8];     /* symbol name */
        int    n_type;          /* type flag */
        unsigned n_value;       /* value */
};


struct bexec {              /* b.out header */
        long  xb_magic; /* magic number */
        long  xb_text;  /* text segment size */
        long  xb_data;  /* data segment size */
        long  xb_bss;   /* bss size */
        long  xb_syms;  /* symbol table size */
        long  xb_trsize;   /* text relocation table size */
        long  xb_drsize;   /* data relocation table size */
        long  xb_entry; /* entry point */
};
```

See Also

masm(CP), ld(CP), nm(CP), strip(CP), xlist(S)

Value Added

x.out is an extension of AT&T System V provided by the Santa Cruz Operation.

xbackup

XENIX incremental dump tape format

Description

The *xbackup* and *xrestore* commands are used to write and read incremental dump magnetic tapes.

The backup tape consists of a header record, some bit mask records, a group of records describing file system directories, a group of records describing file system files, and some records describing a second bit mask.

The header record and the first record of each description have the format described by the structure included by:

#include <dumprestor.h>

Fields in the *dumprestor* structure are described below.

NTREC is the number of 512 byte blocks in a physical tape record. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE Tape volume label.

TS_INODE A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.

TS_BITS A bit mask follows. This bit mask has one bit for each inode that was backed up.

TS_ADDR A subblock to a file (*TS_INODE*). See the description of *c_count* below.

TS_END End of tape record.

TS_CLRI A bit mask follows. This bit mask contains one bit for all inodes that were empty on the file system when backed up.

MAGIC All header blocks have this number in *c_magic*.

CHECKSUM Header blocks checksum to this value.

The fields of the header structure are as follows:

c_type The type of the header.

c_date The date the backup was taken.

c_ddate The date the file system was backed up.

c_volume The current volume number of the backup.

c_tapea The current block number of this record. This is counting 512 byte blocks.

c_inumber The number of the inode being backed up if this is of type TS_INODE.

c_magic This contains the value MAGIC above, truncated as needed.

c_checksum This contains whatever value is needed to make the block sum to CHECKSUM.

c_dinode This is a copy of the inode as it appears on the file system.

c_count The following count of characters describes the file. A character is zero if the block associated with that character was not present on the file system; otherwise, the character is nonzero. If the block was not present on the file system no block was backed up and it is replaced as a hole in the file. If there is not sufficient space in this block to describe all of the blocks in a file, TS_ADDR blocks will be scattered through the file, each one picking up where the last left off.

c_addr This is the array of characters that is used as described above.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END block and then the tapemark.

The structure *idates* describes an entry of the file where backup history is kept.

See Also

xbackup(ADM), xrestore(ADM), filesystem(F)

Value Added

xbackup is an extension of AT&T System V provided by the Santa Cruz Operation.